Improving File Navigation with Spatially Consistent Revisitation Visualisation

COSC460 – Honours Project Report November 2, 2012

Joshua Leung
jsl76@uclive.ac.nz

Department of Computer Science and Software Engineering
University of Canterbury, Christchurch, New Zealand

Supervisor: Professor Andy Cockburn
andy@cosc.canterbury.ac.nz

Abstract

People are storing an increasing amount of their data digitally in the form of files. However, many of the current navigation based interfaces are unable to support efficient retrieval of this information. We develop a file system crawler, and use this to conduct a user study to characterise the structural features and temporal usage patterns of file systems. Based on these findings and the extensive prior literature, we develop a spatially consistent representation of entire file systems which is augmented with colour-coded tags allowing efficient access to temporally relevant target folders within the file system (SCOFT). We conduct a user study to evaluate the effectiveness of this technique and the supporting techniques developed. Our findings show that SCOFT allows users to revisit files 3 times faster than when using a standard file browser.

Acknowledgments

The author would like to thank his supervisor, Andy Cockburn, for his guidance, inspiration, and support throughout this project. He would also like to thank his parents for their love and support. Thanks must go to all the experiment participants for volunteering their time and valuable feedback. Thanks to my fellow Honours students for many interesting discussions and an great year, Stephen Fitchett for doing some inspiring work and help with AccessRank, and the rest of the department for interesting questions and interest in my project. And finally, my trusty laptop which I've spent countless long hours of frantic hacking, debugging, pondering, and report writing with along the journey to making the world a better place. Thanks!

Contents

1 Introduction

1.1 Problem Summary

People are storing an increasing amount of their data digitally in the form of files. These represent information such as documents, images, videos, music, and other application-specific data. They are typically grouped hierarchically using "folders", which can contain an arbitrary number of files and can be nested inside each other arbitrarily. This model is known as the Hierarchical File System (HFS) and has been the dominant file system model since being introduced in the 1960's [5].

There is growing evidence that HFS's, or more specifically, file browsing tools such as Microsoft's "Windows Explorer", Gnome's "Nautilus", and Apple's "Finder" are inadequate for supporting the range of activities performed by computer users. Several independent research groups have recently characterised and verified the existence of a number of well known problems with these tools such as inefficient file retrieval times [6] and frequent "navigation failure" (or difficulty locating previous saved items) [60] [6].

File management systems have also been extensively studied, with several different research directions in the literature. Some focus on changing the file and information management paradigms used (e.g. relational [27], and temporal [23]). Others have developed visualisation techniques for hierarchical data structures [45] [35] [24], augmentation techniques which support user activites [2] [20], and hybrid systems such spatio-temporal ones [46] [51]. However, it appears that many of these attempts have not been successful as they were regarded as being too radical for widespread adoption.

Perhaps a more user-centered approach is needed [39]. Recent research has identified some promising design principles for designing more effective and efficient user interfaces by focussing on the human factors. It has been found that users' spatial memory can be used to enable efficient performance for typical computing tasks when item locations are stable and predictable (i.e. "spatially consistent") [13] [50] [49]. Another key observation is that human behaviour is often repetitive, and has strong temporal characteristics upon which many predictive algorithms have been developed [1] [21].

In this thesis, we investigate whether we can create a spatially consistent visualisation of file systems which allow users to quickly and repeatedly access salient target locations. We characterise file system structure and temporal characteristics of real-world file system datasets. The information collected is used to perform preliminary feasibility evaluations of proposed design solutions, and to validate the robustness of prototypes to "real-world" inputs. Finally, we perform a user study to evaluate the usability and performance characteristics of our interface under controlled laboratory conditions.

1.2 Project Goals

This project aimed to investigate how spatial memory and the repetitive nature of human behaviour can be used to improve file navigation interfaces.

Therefore, the sub-goals of this project are to:

- Understand and identify important characteristics of how file systems are used
- Identify the problems of previous approaches/systems
- Determine how an interface can be built to leverage spatial memory and revisitation behaviour
- Evaluate the effectiveness of said interface for increasing efficiency and satisfaction of users for file navigation tasks

1.3 Achievements

This project presents a number contributions to Human Computer Interaction (HCI), and particularly to the sub-field of Personal File Navigation. These include:

- A classification scheme and review of past file browsing/management systems
- An analysis of file hierarchy structures used by postgraduate research students
- An analysis of the temporal distribution of file system contents
- A technique for identifying "software distribution" file hierarchies
- A widget displaying a spatially consistent visualisation of a users' file system, augmented with coloured tags to facilitate revisitation activities (SCOFT)
- Three supporting augmentation techniques for file browsers to improve navigation performance

The supporting augmentation techniques developed were:

- Temporal Search and Item Highlighting
- Hierarchy Flattening – Navigation shortcuts for folder chains
- Crabbing – Navigation to siblings (direct siblings and in parallel hierarchies)

Figure 1.1 shows a screenshot of the prototype file browser that we developed, which incorporates all of the techniques described.

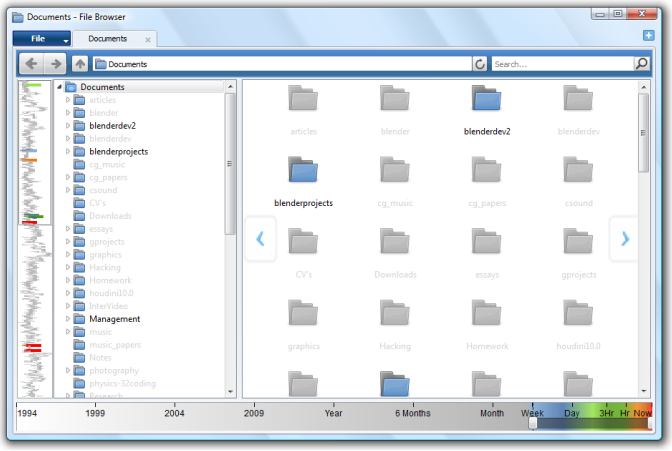

Figure 1.1: *Screenshot of the prototype file browser (FileShell) we developed. SCOFT is the rectangular widget beside the folder tree. A bidirectional time range slider sits along the bottom, and controls the temporal filtering of items. Left/right buttons on either side of the icon view are for 'crabbing'*

1.4 Report Overview

Chapter 1 Discussion of the problem domain, key objectives, achievements, and research approach for this project

Chapter 2 Review of prior systems for file navigation and management

Chapter 3 Discusses the structure, characteristics, and usage of file systems

Chapter 4 Presents the design goals and principles underlying our approach

Chapter 5 Discusses the characteristics and implementation details of techniques

Chapter 6 Presents a user study conducted to evaluate the effectiveness of this interface

Chapter 7 Analysis of evaluation results, and discussion of relative merits/weaknesses of our work

2 Related Work

File systems and hierarchical data sets are used in many different domains and have been the subject of many research projects. Many alternative organisation paradigms, visualisation techniques, and interactive techniques have been developed to improve the usability of these systems. However, the majority of these techniques have not gained widespread adoption in production systems within their their target domains.

In this chapter, we present an structured review of prior approaches, based on their primary characteristics (or limitations) and how they relate to our project aims. An overview of this classification scheme is presented in Figure 2.1.

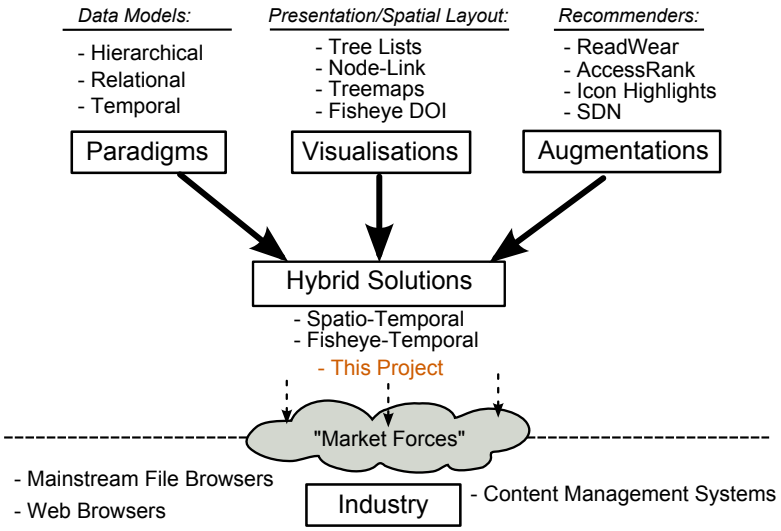

Figure 2.1: *Schematic showing an overview of the classification scheme used here*

2.1 Personal File Management

Personal Information Management (PIM) is an active sub-field of HCI which is focussed on understanding and improving the ways in which people interact with all the data stored on the computers from a variety of sources [14]. Information from different sources such as the file system, emails, contact lists, browser bookmarks, calendar entries, and music playlists are all considered as equal entities, and collectively define the personal information space of a user.

Our work falls within a sub-area of PIM known as Personal File Management (PFM) [6]. As the name suggests, this focuses on understanding and improving the ways in which users interact with their file systems. As discussed in Chapter 3, many of the key issues (such as fragmentation, classification, and revisitation problems) identified by PIM researchers also apply to file systems.

2.2 File System Paradigms

We begin our discussion by reviewing some of the different paradigms for organising and retrieving information. Here, "paradigm" refers to a framework which combines some data model describing how relationships between items are managed and the underlying philosophy behind the interaction style associated with this model.

We review three paradigms in this section: Hierarchical (i.e. nested groups), Faceted (i.e. semantic

tags), and Temporal (i.e. timestamps). The two latter systems were proposed as PIM systems for revolutionising the ways that people can interact with their computers, and were driven by the belief that the cause of all information management problems was the underlying data model in use.

2.2.1 Hierarchical File Systems

Hierarchical File Systems (HFS) are the dominant method for organising files, and have been used since they were introducted in the 1960s [5]. This can be attributed to the simplicity of the underlying metaphor. Files represent blobs of data (such as a different types of documents), and are grouped together inside folders. These folders can be nested inside each other to form a hierarchy of folders, which acts as a categorisation scheme for the files they hold.

The entities here correspond directly to those in traditional "paper" offices, where documents (i.e. pages of information) are grouped together in folders, which are then stored in elaborate filing systems (bookshelves and filing cabinets) [41]. This simple conceptual mapping would have appealed to early adopters and managers looking to introduce computerised systems to the organisations. However, many of the problems and limitations associated by these systems (as noted by Maloney [41]) also affect their computerised equivalents [60]. In particular, users of both systems frequently have problems remembering where items of interest are stored within these systems, even when they created the system themselves [60].

Various authors suggest that these problems are caused by the restrictive structure imposed by tree-based information systems [60][4]. They claim that hierarchies are ill-suited for categorising information, as items can belong to multiple categories at a time, while the hierarchy will only allow them to be grouped under a single category [60]. Attempts to work around this problem by nesting the folders inside each other often make this problem worse, as the classifiers can be used in many different orders. It has been suggested that this ambiguity is a possible cause for navigation failure and other revisitation problems, as well as increasing cognitive load when deciding where to save a file [4].

2.2.2 Faceted or Relational Systems

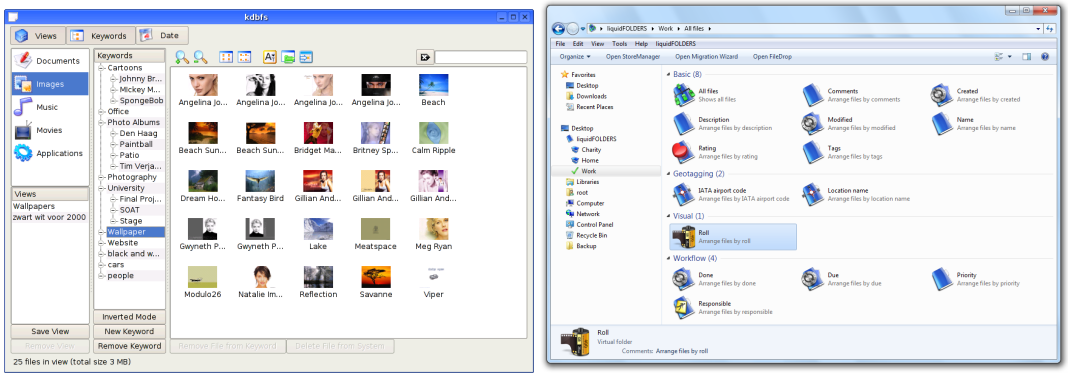

Figure 2.2: *Examples of Faceted File Systems. These are available for download from their respective websites (from which these images were obtained). 1) KDBFS [28], 2) LiquidFolders [40]*

Faceted (or alternatively, tag-based or relational) systems were proposed to avoid the classification problems of HFS. Figure 2.2 shows examples of free and commercial implementations available on the internet. This class of systems focus on allowing users to model the semantic relationships between files by decoupling file location from their classification information, as each file can have an arbitrary number "tags" or category labels attached as metadata instead of just a single parent. [27].

Although users have increased freedom to categorise their information (including a greater potential not to do so [10]), they are forced to use a search-directed interaction style when retrieving files. That is, they need to specify sets of tags every time they want to retrieve their files. However, recent studies have found that users have strong preferences for navigation-based retrieval over search-based retrieval, regardless of search engine quality [5]. The authors of that study suggest that the cognitive demands for search (involving query formation, and recall as opposed to recognition) often outweigh the benefits.

Another key disadvantage of search-based systems is that they typically do not present items in any predictable or spatially stable/consistent order. Instead, the items may be sorted in terms of relevance relative to some arbitrary metrics defined within the ranking algorithms used. Lack of spatially stable item locations means that users must resort to visual search when using these systems, thus limiting performance [5].

2.2.3 Temporal Systems

Unlike the two other paradigms presented here, temporal interfaces are based on the idea that it is easier for users to think of files (or snippets of information) in terms of the relative chronological order in which they interacted with these. The underlying assumption is that users can infer the semantic context of these entities from a combination of the modification time and the items which were modified around the same time.

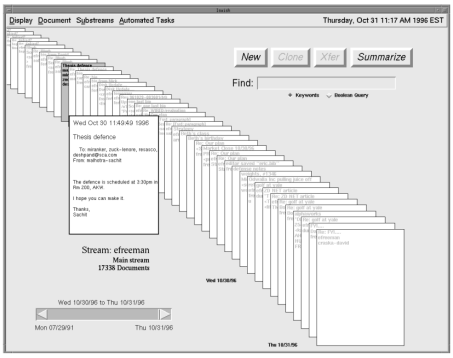

Figure 2.3: *Screenshot of the LifeStreams system, as presented in Freeman's thesis [23].*

This was the subject of Freeman's thesis, which described a "time-ordered stream" paradigm called "Lifestreams" [23] (see Figure 2.3). It was more of a PIM system, as it represented files, emails, and other information snippets within a single unified "stream" that showed each of these items as page-like icons that faded back into the screen. More recent items were displayed closer to the user, while older items were displayed further back. A slider was provided to navigate within this stream to find the approximate time at which items were modified.

A similar concept is used in the "Stuff I've Seen" system by Microsoft Research [14] [17]. However, unlike Lifestreams, Stuff I've Seen is not a pure temporal interface. Instead, it is a search-based system which allows searching items by a number of semantic tags, but displays these results as a time-based list of results.

Neither of these systems has replaced HFS as the dominant file management paradigm. However, the temporal model has been used in other systems such as in social media networks. Most notably, Facebook adopted a "Timeline" view as their default interface [18], which shows all user activity in chronological order in a manner which is very similar to Stuff Ive Seen [14].

2.2.4 Summary of Paradigms

Paradigm change (or rather, radical paradigm change as attempted by these researchers) does not appear to be the right approach to solving the file management and retrieval problems. Hierarchical File Systems are still the dominant file system paradigm in use, and have become even more widespread as they are now used on all consumer devices (such as cameras and smart-phones) [26].

2.3 Visualisations of Hierarchical Data (Spatial Systems)

A parallel research direction in Human Computer Interaction and Information Visualisation focusses on ways of presenting hierarchical data to users. We refer to these systems as "Spatial" systems, as they are primarily focussed on the presentation aspects for hierarchical datasets based on the structural characteristics such as parent-child relationships in HFS datasets. This information is used to determine the shape, layout, and sometimes behaviour of the resulting visualisation such that certain indicators of quality are satisfied. These criteria include the ease of navigation, how aesthetically pleasing the visualisation is, how clear the structural features are conveyed, and whether the visualisation makes optimal use of the screen space available.

The range and diversity of approaches that have already been tried was somewhat surprising. Many of our initial ideas (and subsequent ideas derived from those) had already been unsuccessfully attempted in past studies already in some form. Therefore, analysing these systems proved to be a useful exercise in better understanding the scope of designing spatial representations of file systems.

2.3.1 Standard Tree Lists

Let us firstly consider the interfaces used for hierarchy navigation in standard file browsers. These are: Expand/Collapse, Multi-Pane, and Spatial Windows. See Figure 2.4 for illustrations of each type.

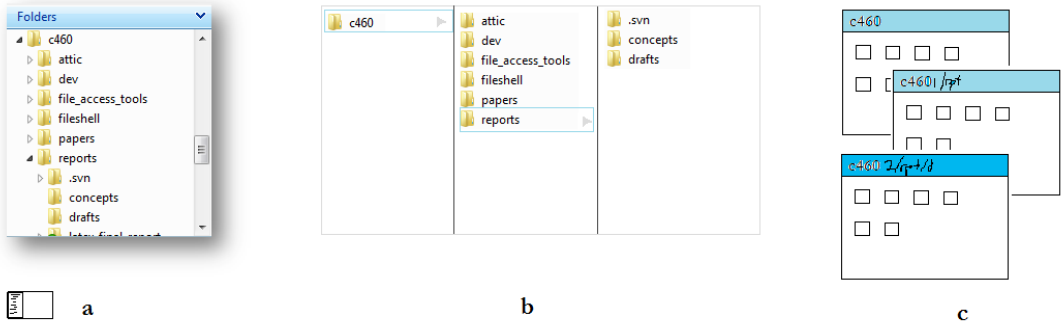

Figure 2.4: *Standard tree interfaces: a) Collapsible, b) Multi-Pane, c) Spatial Windows*

Expand/Collapse

This technique displays the hierarchy in a list-like format, where items are listed out in Depth First Search (DFS) pre-order traversal order. Items are indented relative to their depth in the tree, and can be expanded/collapsed to show/hide their subtrees.

It is a standard widget in all modern user interface toolkits such as Qt [16], and is also the standard folder hierarchy representation in Windows Explorer and Nautilus.

When used for file navigation, the Expand/Collapse tree widget is typically used in conjunction with a "list view" panel to form an Overview + Detail interface (see Figure 2.4a). The tree widget provides users with an overview or contextual information about the hierarchy, while the list view displays the files and folders contained within the currently selected folder.

Multi-Pane

This technique displays each level of the hierarchy in adjacent panels. The contents of panel $n+1$ depends on what item was selected in panel n. Adjacent panels can either be stacked vertically (as per [11]) or placed beside each other horizontally (as in Figure 2.4b). It is one of the three presentation modes that Finder commonly uses.

Spatial Windows

This technique displays each level of the hierarchy as a separate window that can be placed in different locations on the screen by users (see Figure 2.4c). It has been claimed that this provided a better metaphorical mapping to physical folders which can be arbitrarily placed on a desktop/workspace with less spatial overloading [57]. However, it is prone to "window hell" problems (where users become overwhelmed with too many windows open on screen at once and are unable to easily manage them), especially with deeper hierarchies. This used to be found in all file browsers, although this is now restricted to just Finder and Nautilus, and does not appear to be enabled by default in most cases.

Analysis and Evaluations

Chimera and Shneiderman [11] conducted an evaluation involving two of these interfaces, and found that the Multi-Pane interface supports faster hierarchy navigation than the traditional Expand/Collapse interface does, although it does have greater space requirements. However, it is worth remembering how popular and familiar the Expand/Collapse interface is.

2.3.2 Node-Link Approaches

Some closely related approaches are Node-Link visualisations. These display files and folders as "nodes" (typically represented by boxes and circles), with arrows showing relationships between these. Unlike in tree-lists, items in Node-Link visualisations can be placed anywhere on a 2D (or even 3D in the case of Cone Trees [47]) canvas, instead of being restricted to being aligned in a single column.

The differences between these presentation styles reflect the differing underlying design objectives. Node-Link visualisations aim to be standalone tools where all the information is presented as part of a single unified canvas. In contrast, tree-list interfaces are designed to be used as supporting elements which can be embedded within many different interfaces as unifying standard components.

Examples of these interfaces include Cone Trees [47], Hyperbolic Trees [34], and SpaceTrees [45] (see Figure 2.5.

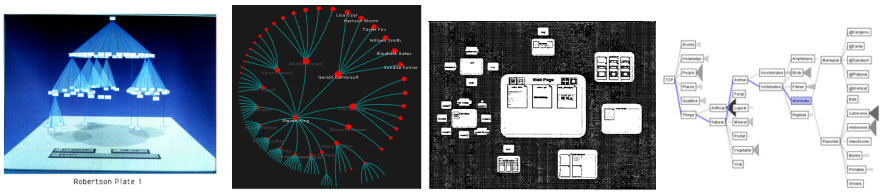

Figure 2.5: *Examples of Node-Link visualisations of file systems or hierarchical data sets: 1) Cone Trees [47], 2) Hyperbolic Trees [34], 3) Gold Leaf [19], and 4) Space Tree [45]*

Analysis of Node-Link Approaches

Node-Link approaches appear to be more focussed on creating aesthetically pleasing graphics for casual data exploration than on maximising user productivity.

Apart from the Space Tree, all of the approaches are optimised for presenting a single level of the hierarchy to the user at a time for visual simplicity. This means that in order to traverse the tree, users must click through each level of the tree to expand it, and select items in the next level. Animated transitions are often used between states and add some delay to each navigation step. However, it can be argued that these are important for helping users maintain spatial awareness.

Space Trees aim to make more efficient usage of the available screen space by automatically expanding or collapsing levels as appropriate so that users can access as many different parts of the tree as can be accommodated. Summaries of collapsed parts of the tree are shown using various visualisation techniques.

Another important limitation is that many of these techniques are typically only illustrated using hierarchies with very small numbers of children, evenly distributed at each level. As discussed in Chapter 3, this assumption is incorrect. Lower level folders tend to have high branching factors, while a small number ("dump") folders can contain several thousand files or more. When used for real world distributions, these techniques often become unmanageable.

2.3.3 TreeMaps

No discussion of file system visualisation techniques would be complete without mentioning TreeMaps. This space-filling technique was first introduced in a seminal paper by Johnson and Shneiderman in 1992 [35]. Folders are represented as rectangles which are recursively nested inside their parents, with various packing algorithms designed for determining "optimal" layouts of these rectangles.

They are an example of the Zoomable User Interface (ZUI) concept, where everything is represented on a single, infinitely zoomable canvas [59]. To see more details about an item, users zoom in on the area of interest to see more semantically relevant information. Proponents of this approach commonly cite the benefits of having a unified information access metaphor [59]. That is, the use of semantic zooming for filtering level of detail displayed.

The original paper launched several new research directions in HCI and visualisation. Researchers have studied the human performance aspects of this technique such as how efficiently people can navigate around the information space [52], and also visualisation quality aspects such as designing optimal layout algorithms for maximising particular aspect ratios [3]. A number of derivative techniques have also been

developed, including using circular bubbles instead of rectangles [9]. Radial versions such as Sunbursts [53] also exist.

Analysis of Treemaps

There are two primary limitations with this approach. Firstly, they do not appear to be well suited to supporting efficient revisitation behaviours, as users still need to perform several navigation steps to reach targets which may be nested several levels deep. For example, a user may need to perform several zooming and panning operations to be able to find a frequently accessed target located four levels deep within a subtree containing just a few children, but located within a file system that also contains several large sub-hierarchies (e.g. collections of papers or former projects).

Another limitation is related to the unconventional presentation of file systems using this visualisation technique. This problem is especially pronounced now, as many computer users have developed extensive experience working with hierarchical file systems presented using traditional techniques. To be productive with this system, users are forced to become familiar with an alternative visual model for representing file systems. However, there are few visual cues that help users make this association, while the often irregular placement of labels makes quickly scanning the information space more difficult [38]. For many users, potential performance gains are offset by the considerable retraining effort required, thus diminishing their general appeal.

2.3.4 Analysis of Spatial Techniques

Overall, spatial interfaces have a number of problems. Apart from standard tree lists, most are space-intensive as they were designed to used as standalone tools. Thus, they are unsuitable for being embedded into other tools (for example, as a project browser within an Integrated Development Environment) or for comparing the contents of two directories to see which files may be missing. Many also have scalability problems, as they can only be used on small academic "toy" problems with artificially generated datasets, and are unusable on production systems.

Another limitation of spatial interfaces is that they do not facilitate efficient revisitation behaviours. In all of these systems, users need to perform multiple navigation steps ranging from opening/expanding multiple categories, to scrolling through a long list of items. This is inefficient for accessing items that were recently or commonly accessed. However, such behaviour is acceptable when facilitating data exploration.

One problem which may arise when performing data exploration is that standalone spatial visualisations can only provide users with a single level of detail at a time. For some tasks this limitation does not cause any problems. However, if users become spatially disorientated or need to be able to understand the relationships between items in different areas of an information space, then the inability to view the surrounding context can be problematic.

2.3.5 Fisheye Degree of Interest

Fisheye visualisations are distortion-based "*focus + context*" techniques which aim to avoid these problems. They do so by taking a detailed view of the area immediately surrounding a target ("focus") and integrating this into an overview of the entire information space ("context") to show the approximate location of the focus zone. We do not include these interfaces under spatial interfaces, as unlike spatial interfaces, they do not temporally separate different states or require additional supporting views ("overview + detail") to allow users to reason about the relationships between various items.

The seminal work in this area is Furnas' 1986 paper, "Generalised Fisheye Views" [24], which defines a framework for implementing fisheye views using a two-pass technique. In the first pass, a function called the "degree of interest" (DOI) function is computed for every item in the visualisation to determine its relevance. The second pass uses these values to distort the information space so that items with higher scores are given proportionally more space, while those with lower scores are suppressed and potentially moved away from their original locations to free up space for more relevant items.

Although popular implementations of fisheye visualisations typically define cursor-dependent "fisheye lens" distortions, this framework also allows data-dependent distortions which do not depend on cursor location and thus provide slightly better spatial stability and predictability for target acquisition as the visualisation is at least stable until the dataset changes again. An example of this is [51], discussed in a later

section. Other examples of fisheye visualisation techniques include [56] and [36], which investigate different techniques for defining Fisheye DOI functions for hierarchy navigation with a focus on file systems.

2.4 Assistive Augmentations

We now focus briefly on a few techniques which have been developed to provide support for navigation activities within the frameworks offered by standard systems. Variants of these techniques have been adopted in commercially available systems. The techniques we describe in Chapter 5 are directly inspired by some of these systems.

2.4.1 Summary Visualisations

Read Wear scrollbars [32] are a technique for augmenting scroll bars with revisitation information (see Figure 2.6a). In the original paper, Hill and Hollan superimpose a frequency distribution over scroll bars based on how frequently information was edited on each corresponding line in the document. They claim that this allows users to see which parts of documents are most stable or conversely most frequently edited, thus supporting revisitation of areas of high activity.

Alexander et al. [2] extend this concept based on the results of a log-based study of revisitation behaviours for document editing. Instead of showing the frequency distribution, they display a number of colour-coded tags, with labels indicating the order in which they were last visited.

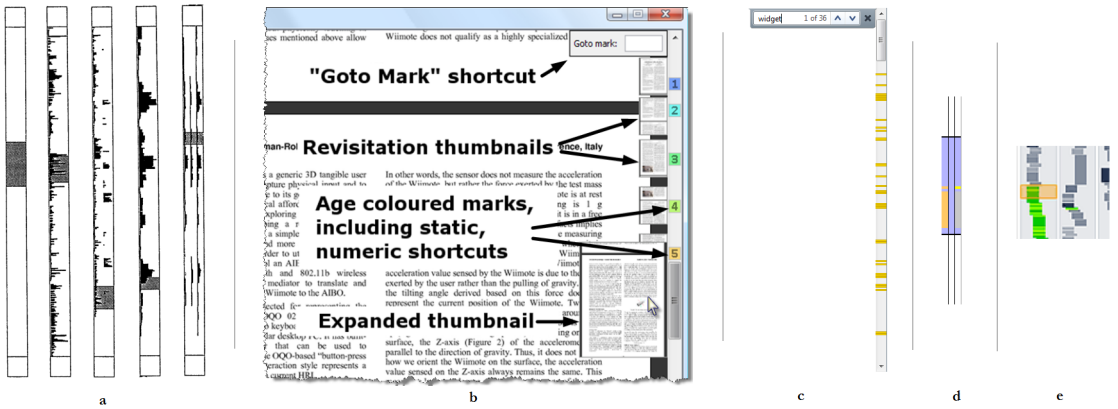

Figure 2.6: *Examples of summary visualisations displayed on scrollbars: a) Read Wear [32], b) Footprints [2], c) Google Chrome search matches [25], d) TortoiseDiff "change summary", e) ForumReader [15]*

Similar techniques have been used in commercially available software. Most notably, Google's *Chrome* Web Browser [25] highlights locations of search matches on the scrollbar showing matching items. Apart from highlighting items of interest, researchers have also used this to visualise and support navigation in hierarchical datasets such as online discussion forums [15], where each post is visualised as a block.

2.4.2 Selective Highlighting

Fitchett and Cockburn [20] developed a number of techniques for improving file navigation. The primary contribution of this paper is the use of ambient visualisations (in the form of icon highlights) to highlight and guide users towards salient targets based on user-provided keywords ("search directed navigation") or based on previous interaction history using AccessRank [21]. A secondary technique presented in this paper is "hover menus", which allows users to navigate through hierarchies more efficiently by allowing users to jump to folders that they've previously visited (again using AccessRank for predicting the relevant targets). These techniques were shown to be reduce the time taken for target revisitation, both individually and in combination.

2.5 Hybrid Systems

Finally, we will review a number of systems which have combined various organisation paradigms and visualisation techniques to take advantage of the relative benefits of each. As these maintain many familiar aspects from existing systems, users are able to adapt to the new systems with less effort.

2.5.1 Spatio-Temporal

Spatio-Temporal systems allow users to inspect snapshots of their file layouts at different points in time. They are designed to allow people to leverage their spatial memory as items are kept where users left them at certain points in time. However, they also recognise that information may only be relevant for certain time periods, so they provide time browsing capabilities to navigate through different temporal snapshots while maintaining the semantic context which may be associated with that space-time combination.

We found two examples of these systems: Timescape [46] and TimeSpace [37]. Both focus only on representing a single-level desktop workspace, where users can freely arrange their files as desired. Temporal browsing capabilities allow users to browse though different states, with items appearing and disappearing as fade in and out of relevance.

2.5.2 Fisheye-Temporal

Somewhat related to Spatio-Temporal interfaces are Fisheye-Temporal interfaces. Unlike Spatio-Temporal interfaces, these maintain the traditional hierarchical data structures, but introduce temporal filtering as inputs to the Fisheye DOI functions used for controlling the visibility of items.

One such system is FlexView [51]. This consists of an interface where a pair of sliders can be used to specify the time range where items should be shown in the hierarchy visualisation above. It is also worth noting that a custom tree visualisation technique is used, involving curly braces instead of the traditional line and text model.

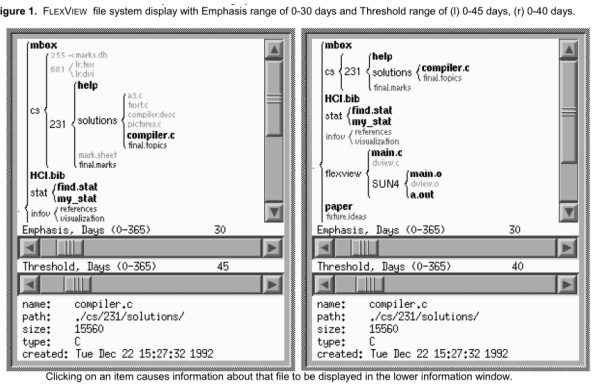

Figure 2.7: *Screenshot of FlexView from the original paper [51].*

3 Characteristics of File Systems

As part of a user-centered design process, it is necessary to understand the domain and characteristics of the system domain [39]. In particular, we were interested in answering the following questions about how current file systems are used:

1. How large and complex are they?
2. What problems do users have when using these?
3. Are there any recurring structural and usage patterns? If so, what are they, and are there any ways in which we may be able to use this information to design better tools?

3.1 Prior Literature

We conducted a literature review to determine what is currently known about the usage of file systems. Three separate groups led by Bergman [5][6] [4][7], Henderson [31][29][30], and Jones [60] respectively, have been actively researching this topic over the past 5 to 6 years. Compared to earlier studies such as [10], these researchers have independently obtained similar results. This may be due to the larger sample sizes and greater consistency between analysis techniques used. However, it is also possible that file system usage habits have evolved as people have gained more experience using them since the earlier studies were first published [31].

From these studies, we identified a number of commonly used metrics for characterising the size and complexity of file hierarchies:

Aggregate Statistics such as total number of files and folders in the entire hierarchy, but also at each depth.

Branching Factors or number of sub-folders each folder contains. This is non-recursive (i.e. descendants of these sub-folders are not included).

Folder Size or number of children (files) each folder contains. This is non-recursive (i.e. files of sub-folders are not counted).

Depth of Nesting for the deepest nested file/folder

Average Target Depth or the depth in the hierarchy where typical targets reside

3.1.1 Structural Parameters

File hierarchies tend to be shallow and broad, with small and well defined folders [6][31]. Average values for these parameters are presented in Table 3.1. It has been suggested that this behaviour "evolved" as people became more familiar with the problems arising from using deeply nested folders [6], and that the tools provided by operating systmes can affect this [7].

Parameter	Average Value
Average Target Depth	3
Depth of Nesting	10
Branching Factor	10
Folder Size	11

Table 3.1: *Structural parameters of file systems, as presented in prior literature*

3.1.2 Classification Behaviours

The seminal work of Jones et al. [60] found that users create folders to represent their understanding of *projects* they are working on. Nested folders were therefore a mechanism for representing tasks or sub-projects which required a bit more organisation. This view is also supported by studies conducted by Bergman et al. [4] and Henderson [29].

Jones et al. [60] also found that users tended to have "project templates". That is, they develop standard folder layouts that are then used to structure every project, often by simply copying an existing project and removing the content. We believe that this implies that there better tools could and should be developed to make use of these recurring structural patterns, thus helping users to navigate and manage their files more effectively. This may also be the cause of the "shallow and broad" folder distributions, as users are simply creating "subject folders" containing many separate "project folders", which are all based on the same underlying template structure.

3.1.3 Common Problems

Hierarchical organisation schemes are not suited for classifying items for which multiple classifications are possible [60]. As multiple classifications are possible, there are often ambiguities about how an item may be classified, leading to confusion. [4]. This is known as the project fragmentation problem, and results in parallel hierarchies [4][30]. As discussed earlier, faceted file systems [27] were developed to overcome these problems.

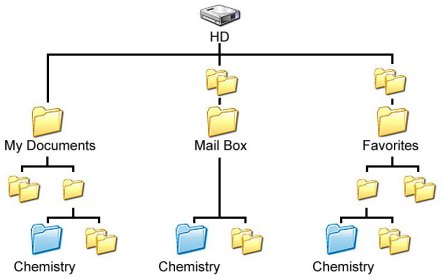

Figure 3.1: *The parallel hierarchies (or project fragmentation) problem. Image from [4]*

Due to the project fragmentation problem, users frequently have difficulty remembering where their files are saved [60]. This is known as the *navigation failure* problem [10]. In one study, it was found that only 94% of navigations to *recently used* targets were successful as users had no idea where some files were located would give up trying [6]. A significant number of retrievals were deemed "eventually successful" as users only had a vague idea of where those were located and had to try several times to successfully locate those files. Finally, there was other other category of files, where users knew exactly where the target was located as it was a target they had recently accessed very frequently (e.g. main file for a report). They were able to effortlessly navigate to these targets without much thought [6], as they were able to use their spatial memory instead of relying on visual search and making navigation decisions at each level [12].

Another problem that has been identified is that hierarchy navigation is inefficient. Bergman et al. [6] found that on average, it took users 14.76 seconds to navigate to targets that had been recently accessed.

3.2 File Crawler – User Study

In addition to performing this literature review, a user study was conducted to verify the findings identified and to provide greater insight into some aspects which were not well covered in the literature.

3.2.1 Apparatus

A "file crawler" Python script was developed to traverse folder hierarchies starting from a nominated 'root' folder or folders. These root directories were generally the user's "Home" directory (for example, `/home/cosc/students/jsl76`) or an equivalent if the majority of their files were stored elsewhere (for example, `C:\Courses` in the case of one participant). The directories selected for traversal were chosen in collaboration/with assistance from study participants.

Information Capture

This collected information such as the names of files and folders, the nesting relationships between these (such as which folder is the parent of what folders and files), timestamp metadata (creation, modification, and access times), and the usage of shortcuts within the hierarchy.

All of this information was written to a plain-text file with a custom format (see Appendices). This was done to facilitate easier parsing and manual filtering without the overheads associated with using some other more formalised structures (such as character sequence escaping, and maintaining parsing state information). For example, different types of information was indented differently for easier extraction and processing, while JSON snippets were used for representing file/folder metadata to be more tolerant of error conditions and system variations.

Omitted Information

In order to protect the privacy of study participants, the crawler did not save information about files or folders marked as "hidden", and would not traverse the subtrees rooted such folders.

The crawler also ignored files and folders whose filenames started with a period. On Unix/Posix systems, such filenames are traditionally considered to be "hidden" files and folders [44]. These also tended to be system files, such as configuration settings files (`.recently-saved-hist` or application metadata (`.git/`, `.svn/`). Such files are generally unlikely to be of much semantic interest to users, and were thus omitted from the traces.

Issues Encountered

Although charset and encoding issues were initially overlooked, these caused problems during later stages of the processing and analysis pipeline developed. In particular, this caused problems when trying to convert these datasets to Sqlite3 databases (to improve dataset loading times, which ranged from a few 2-3 seconds for a relatively simple dataset, to over a minute on some of the more complicated datasets). Consequently, all datasets generated needed to be manually sanitised (i.e. by saving over the relevant files using a text editor, after choosing UTF-8 encoding).

There were also a number of other unanticipated situations which caused the crawler (and subsequent parser) to fail in different ways for the initial few datasets collected. These included problems with broken symlinks, a file without a name (i.e. it's name consisted of a single space), and a folder with a backslash for a name.

It was also somewhat irritating to discover that the "`ctime`" attribute has a different meaning across Windows (where it is "creation" time), and on Unix-based systems (where it is the last time when the file properties changed), with no suitable alternatives for obtaining this information. This had consequences for some of the processing and target identification work performed later.

3.2.2 Participants

There were 8 participants in this study. Participants were chosen on a semi-random basis from the department, based on their availability during the data collection period. Most were post-graduate students who had been studying at the university for at least 3 years, but had only recently started their coursework for the year. Hal of the file systems crawled were "home" directories on departmental Linux accounts (5 systems), while others were from personal devices (4 Windows, 1 Mac).

3.2.3 Results

Table 3.2 shows the variation in dataset sizes in this study.

Parameter	Average Value	Median Value	Low	High
Total Number of Files	45734	9569	199	300113
Total Number of Folders	7471	1222	245	13053

Table 3.2: *Size of datasets. Note that majority of datasets in this study tended to be smaller.*

We found that most folders tend to be occur closer to the root (i.e. hierarchies are mostly shallow), with most folders between 4 and 6 levels deep (see Figure 3.2).

When visualising the structural parameter distributions for individual datasets, we found that it was often necessary to log transform the graphs, as they tended to be heavy tailed distributions. There would usually be a large number of folders with very low values for each structural parameter, with one or two

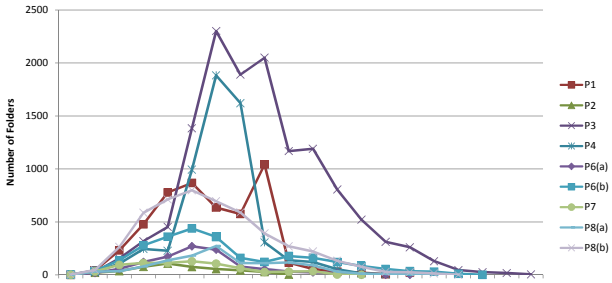

Figure 3.2: *Graph showing the distribution of folders at each level for each participant. One dataset has been excluded from this analysis as it was much larger than any of the other datasets.*

folders that had very large values. Folders which large values like this were usually big unsorted directories ("dump" folders). As can be seen in Figure 3.3, folders at shallower depths tended to have structural parameter values (i.e. they were larger), whereas those deeper in the tree tended to be more well-defined.

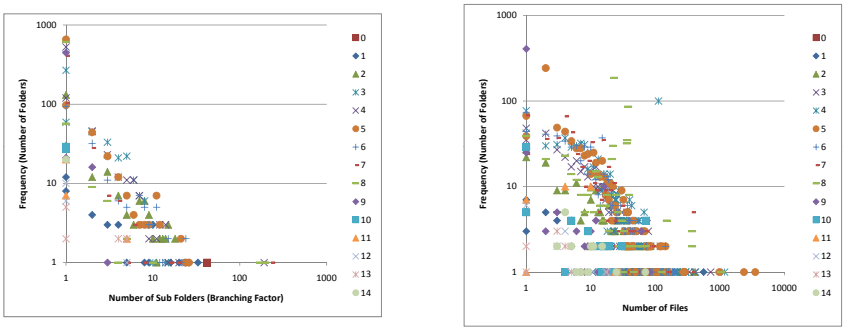

Figure 3.3: *Graphs showing distribution of frequencies (y-axis) of structural parameter values (x-value) for folders at different depths (marker type). Left: Branching Factor, Right: Size (Number of Files) Factor.*

3.3 Visualisation Experiments

We attempted to visualise these datasets using the GraphViz Toolkit [33] as an exploratory exercise. This was done in order to:

1. Determine the suitability of simple/traditional graph layout algorithms for visualising file system structures

2. Be able to visually identify interesting trends or features in the datasets which are not readily apparent when performing simple statistical analysis of the data distributions

3.3.1 Setup

We used a Python script to convert the file system datasets collected earlier to GraphViz graph description language representations (`.dot` files) using a code generation approach.

Initially, every folder and file became a node in the corresponding graph. File paths were used as the ID values for the nodes to ensure uniqueness. Item names were used as the text displayed on each node in the graph. Parent-child relationships between items were shown as directed links (arrows) between nodes.

We found that GraphViz was unable to cope with such graphs, as there were too many items involved. The layout of the resulting graphs (if they could be created without GraphViz silently terminating without producing any output) were unusable. For example, the `dot` layout for one participant's home directory had dimensions of 32767×19 pixels when all files and folders were included (this was scaled down as the full-sized image could not be handled by any image libraries). This really emphasizes the "shallow

and broad" findings from prior literature. As a result of this finding, we removed files from GraphViz visualisations for all subsequent visualising attempts.

3.3.2 Node-Link Diagrams (dot)

Dot is a graph layout algorithm for directed graphs which can be drawn as hierarchies [55]. This seemed to be the most natural approach, given that we commonly think of hierarchies in terms of top-down trees such as in Figure 3.1.

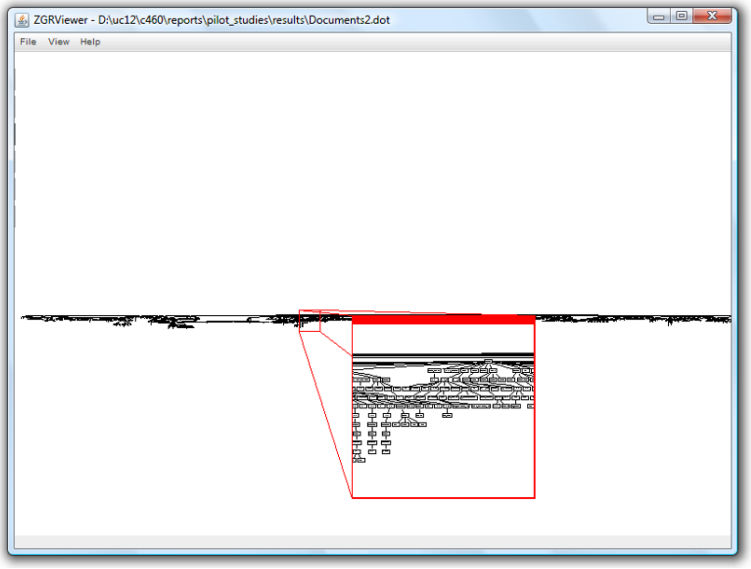

Figure 3.4: *Node-Link visualisation of file system dataset (n = 2188, max depth = 14) using Dot. An inspector lens shows a close up view of part of the hierarchy.*

Figure 3.4 shows a screenshot of the file system snapshot being visualised using Dot. This is the dataset that was mentioned before, except with all the files so that GraphViz could cope. As can be seen, the visualisation is still very shallow and broad (as discussed in the prior literature). When zoomed out to see the entire hierarchy, the structure is completely illegible as no significant features can be seen. Even when just a subset of the tree is shown (i.e. in the inspector lens floating view), there is still too much information presented to be able to understand what the visualisation is showing.

3.3.3 Neato

Neato computes graph layouts using a spring based model, which is solved using an iterative "energy minimisation" algorithm [55]. According to the GraphViz documentation [55], this implementation is only suitable for small graphs containing at most several hundred nodes. Force directed algorithms (such as `fdp`) should be used for larger graphs as they are more scalable. However, we found that Neato produced more aesthetically pleasing results. That is, the resulting visualisations were more compact, balanced, and highlighted structural characteristics more clearly.

The resulting graphs tend to have a radial shape (similar to Sunbursts [53]), with clearly visible banding (see Figures 3.5 and 3.6) resembling the growth rings of trees.

In both of Figure 3.5 and 3.6, we notice the existence of folder chains. Upon closer inspection, we found that these were source code bundles for Java-based projects. In Figure 3.7, we can see that over half of all folders consisted of folder chains corresponding to OSGi project bundles.

From these visualisations, we can see that file systems tend to contain distinct folder clusters at varying depths and with different branching factors (i.e. the radial spread of each cluster). These clusters are most obvious when seen from a "zoomed out" perspective. This suggests that they may be able to act as landmarks which help users to be able to navigate through the hierarchy. However, the space and legibility problems present probably pose problems for end users using this for actual file system navigation.

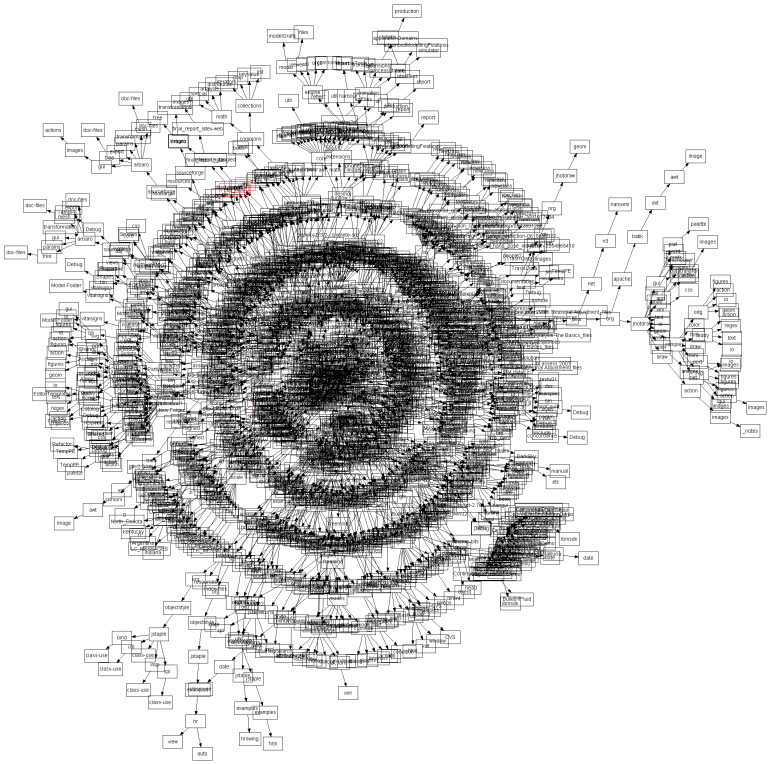

Figure 3.5: *Neato rendering of one dataset (n = 2188, max depth = 14). Note how the folders at different depths are arranged in visible depth bands. Most appear to be about 3 to 4 levels deep as those bands are darker than others further out. Also note how many of the outer folders form long chains (i.e. several nodes linked together, each with only one child) with clusters of folders at their tips.*

3.3.4 Conclusions

Traditional node-link diagrams, especially those laid out using standard graph layout algorithms such as those implemented in GraphViz are unsuitable for visualising real world file systems. Although they are suitable for small academic toy examples, there are serious scalability issues which mean that they are not suitable for production use.

Computing layouts for these datasets was often computationally intensive, and would often end in failure or with ugly glitches. Only some of the simplest datasets (with less than 2000 folders) were able to be visualised, and would take between 15 and 30 minutes on machines with 3 GHz processors. Other more complex datasets (4000 folders) would just mysteriously fail without warning, after 30 to 60 minutes of sustained computation while requiring 6 GB of RAM. In both of these cases, GraphViz only ran on a single core despite being these experiments running on multi-core machines. We are unaware of any any options that would allow multi-threaded execution of rendering jobs.

The other problem with these visualisations is that they are visually overwhelming. Many elements often intersected and overlapped each other, while text was reduced to illegible blobs of pixels (see Figure 3.6). This made it difficult to use these visualisations for file system navigation, where semantic context (or file and folder names) need to be visible. Despite this, when successfully generated, these visualisations can be used to illustrate some representative high-level structural patterns (as per 3.3.3) which are not able to be identified by just studying the statistical distributions.

Figure 3.6: *Neato rendering of a much smaller dataset (n = 444, max depth = 10). Note how depth bands are still visible in some areas, but are less pronounced in some areas (notably the top left corner). Interestingly, folders at certain depths appear to be tightly clustered, suggesting the existence of several distinct project hierarchies. These dense clusters are all preceded by one or more sparse layers, which is suggestive of the existence of folder chains again (see top and bottom clusters).*

Figure 3.7: *A "Downloads" folder containing many Java project example snippets. Note how these form long "folder chains" which radiate outwards, and have dense clusters of folders at their tips.*

4 Interface Design

This chapter briefly discusses the relevant human factors that we took into account when designing our systems.

4.1 Design Principles and Goals

The following core design goals were identified from a number of sources such as problems commonly cited in prior literature, usage and structural patterns of real-world file systems analysed, and our own experiences working with file browsers.

1. Users should be able to navigate to target folders (i.e. frequently visited locations) with as few navigation steps (e.g. clicks and scrolling) as possible
2. Revisitation of items should be encouraged and supported by the system
3. The interface should be similar to standard file browsers to minimise the initial learning burden for easy adoption
4. Spatial stability is needed for users to be able to leverage spatial memory
5. The system should be functional without requiring extensive intervention and setup by users

Many of these goals are interrelated, and are based on commonly accepted usability principles. For example, users develop a spatial model (4) of their file hierarchy based on the representation used/shown by standard file browsers (3). Hence, both of these are necessary to encourage adoption by preserving transferability of past knowledge [43].

The motivation behind the last design goal is that most users do not create shortcuts, even when these would improve their workflow and productivity. As part of our file crawler study (and also during the interviews we conducted at the end of the experiment), we found that most users do not create any shortcuts. Some of the factors contributing to this phenomenon are that users often do not have a priori knowledge of what they are going to need to access frequently in future, and that they are also unlikely to want to spend time managing shortcut lists developed for older projects either Instead, if shortcuts are created they generally point to folders where a large number of their projects are located, as this minimises the costs of maintaining the set of relevant shortcuts.

4.2 Spatial Memory

Spatial memory refers to the ability of humans to remember the locations of items within an information space. This has been found to be robust to linear transformations of an information space [49].

4.2.1 Interfaces Using Spatial Memory

Recent work in Human Computer Interaction has been studying how it can be used to build interfaces which are more efficient to use. For example, CommandMaps [50] and AppMap [48] are two examples of interfaces where commands are laid out in spatially consistent locations. CommandMaps display all the tabs of a Ribbon at once, thus avoiding the spatial overloading problems and associated mode errors which standard Ribbon interfaces have [50]. AppMap is a novel experiment which assigns different clusters of tools in an application to a world map, with each continent representing a different cluster [48].

Spatially consistent interfaces have also been used for improving window switching. SCOTZ [54] is a window switching tool which lays out windows in a spatially stable positions, allowing users to be able to move their mouse in predictable ways to change between different windows.

Interfaces leveraging spatial memory have also been shown to improve user performance for document navigation tasks. Space Filling Thumbnails [13] remove the need to scroll through many pages to reach a target page as all pages in the document are displayed at the same time. This allows users to navigate to any page with just a single click.

4.2.2 Principles for User Interface Design

All of these examples suggest that if the location of items in an interface remain spatially consistent (i.e. they can always be found in a particular place), then the user interface behaviour is predictable, and users are able to rely on this fact to learn that by click in a certain place, certain actions will be performed. That is, spatially consistent interfaces allow users to develop their spatial memory and muscle memory for activating tools quickly by performing a predictable action.

Recent work shows that such interfaces can be distorted, as long the relative locations of points within the distorted interface do not change with respect to each other and the frames of the interface [49].

4.3 The Nature of Revisitation

Human behaviour is repetitive, and can often be modelled using Zipfian distributions [58]. In the context of file system usage, that means that only a small number of items are accessed or used very frequently (and are hence "relevant" to a user), while the majority are not relevant.

4.3.1 Relevance Criteria

In order to be able to help users quickly navigate to targets that they are interested in, we need to be able to determine what those targets are.

Alexander et al. [2] conducted a log based study of navigation within files, and found that revisitation targets can be obtained by using short recency lists of the destinations where users have been recently. Other more advanced predictive algorithms also exist, such as Firefox's Places Frecency [1] algorithm and AccessRank [21]. However, these techniques require a considerable amount of log data before they can begin to produce useful predictions. In our case, we didn't have such data, and attempts to synthesize such "seed" values from the metadata timestamps on the files proved to be a fruitless exercise.

Personal Information Management researchers have also been studying this problem. One study investigated what attributes users remembered about their files [8]. This found that time of last usage, location, file name, and associated events and visual elements were remembered the best. However, they also note that times were only partially remembered, and recommended allowing people to specify approximate ranges either relative to a certain point in time (e.g. "October 2012") or relative to the current time (e.g. "three months ago").

This approximate technique to performing time filtering somewhat resembles the dual sliders in FlexView [51]. These were used to control the size of a time range window that as input for the Fisheye DOI filtering function they were using. It should also be noted that they only used the modification timestamps on the files for performing this filtering. Given the similarity of their approach with our own, it was decided to use the modification timestamps on files and folders in conjunction with a temporal filter for selecting salient targets.

5 SCOFT and FileShell

This chapter discusses the techniques which we developed for improving file navigation. These have been implemented in a prototype file browsing application known as FileShell, which was built in Python 2.6 using the PyQt widget toolkit. The techniques we describe here try to be easy to integrate into existing file browser interfaces. They involve adding a few additional widgets alongside familiar browser components and attaching a few additional custom behaviours to existing widgets to provide additional support.

5.1 SCOFT

Our primary contribution is a widget known as "SCOFT" (**S**patially **CO**nsistent **F**older **T**humbnail). This allows users to quickly jump to any folder in their file system by clicking on the widget near the relative position of that target.

5.1.1 Basic Visualisation – Hierarchy Thumbnail

SCOFT displays an overview of the file system by rendering each folder as a horizontal line (see Figure 5.1). Each of these lines is indented by an amount proportional to the corresponding depth in the tree. The length of each line in pixels is calculated as

$$L_{pixels}(f) = \frac{len(f.name)}{\max_{i \in FS}[i.depth + len(i.name)]} \times W_{SCOFT} \tag{5.1}$$

where f is the file that we are drawing, FS is the set of folders in the file system, and $len(str)$ gives the number of characters in a given string. That is, we display a line whose length is proportional to the largest "name length + indent" value, which is then scaled by the width of the SCOFT widget (W_{SCOFT}) to transform it into "widget space". To ensure that folders with short names are visible, we set a lower bound on line lengths of 2 pixels. All lines are packed/compressed vertically such that they can all fit within the vertical space available in the widget.

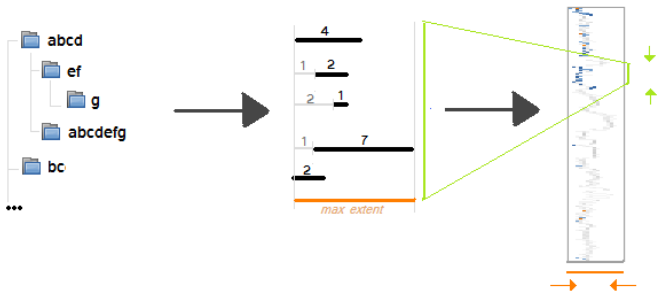

Figure 5.1: *Diagram showing how the folder hierarchy thumbnail is constructed: 1) The folder hierarchy to be visualised, 2) Folders are converted into a series of lines, where each line represents a folder, 3) Lines are packed and remapped to fit within area of widget (i.e. "widget space")*

This was originally intended to be the first step in developing the a Fisheye Degree of Interest (DOI) based file system visualisation. Upon further investigation, we realised that fisheye techniques conflict with our goal of maintaining spatial consistency, even if the DOI function used did not rely on the cursor position. For example, several large by separate clusters of salient targets at the bottom of the visualisation may need more space than is available at the bottom of the visualisation, and would be forced upwards

towards the middle of the widget. This would thus result in the loss of spatial consistency, as item locations can become unpredictable.

The design of this visualisation was influenced by our findings in Section 3.3. We used a vertical representation of file systems as we found that name length variations could result in aesthetically unpleasant layouts in some situations, while making it more difficult for users to efficiently scan for items. The other consideration is that we decided to provide an overview of the folder hierarchy only, while excluding individual files, so that the visual complexity would be more manageable for users.

5.1.2 Temporal Relevance

The basic visualisation provides an interesting perspective of file hierarchies, as it allows users to identify many structural patterns which may not otherwise be noticed. Some of these could also act as salient landmarks. This lead us to wonder whether the temporal distribution (i.e. when various folders and their files were last modified) of folders could also be a useful navigation feature. In particular, revistiation behaviours could be supported by this if it were possible to colour code the folders that had been modified most recently, thus providing additional hints to users.

Our first prototype used a greyscale mapping of time to item colours (see Figure 5.2b). However, we found that it was difficult to accurately distinguish between the various shades of grey. One problem is that the range of time values were often unexpectedly large and would often span over a decade due to a small number of really old files that users may have downloaded. To solve this problem, we mapped time ranges to discrete colour bands (see Figure 5.2c).

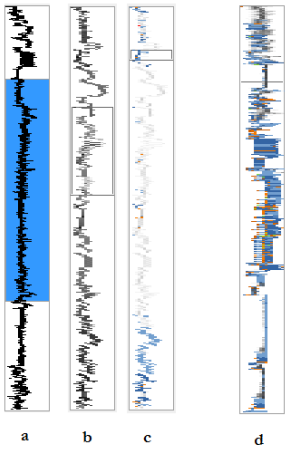

Figure 5.2: *Evolution of SCOFT visualisation. a) First version (no temporal colouring, but with a subtree selected), b) Grayscale, c) Same dataset with basic temporal colouring (time values outside the time range faded to white), d) A larger dataset; time values outside the time range do not fade to white now as that is not legible on some monitors.*

5.1.3 Revisitation Tags

Our original objective when developing SCOFT was not to just develop a static visualisation of file systems, but rather, to develop an interactive component which could be used for navigating the file system. In particular, we needed a way to allow users to quickly navigate to frequently or recently visited target locations.

To achieve this, we over a set of tags over the thumbnail allowing users to jump straight to the corresponding location in the file system. Tags are placed on the thumbnail at the relative location of the target in the file system. They are drawn as coloured lines which are thicker and longer than the lines used to form the hierarchy thumbnail (see Figure 5.3).

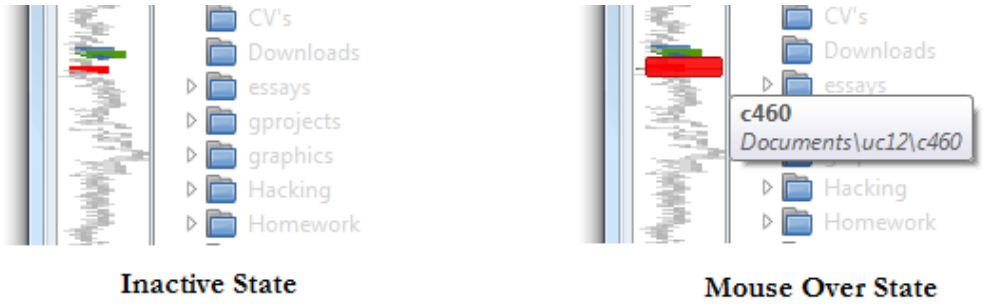

Figure 5.3: *Revisitation tag appearance and behaviour*

Method

1. Divide the widget into a number of "buckets", where each bucket corresponds to one row of pixels in the widget
2. For each item, we add it to 10 buckets above and below the current bucket. The current bucket is updated every n/h items (where n is the number of items, and h is the number of buckets).
3. For each bucket, the item with the largest value within the required time range is set as the current target
4. A bucket's target is marked "important" (i.e. it is marker with a prominent marker to make it easier for users to click on it) if the target in the previous target wasn't marked as important as well. This ensures that targets are not placed too close to each other, making it hard to select and click on one.
5. Items with the "important" flag set are drawn as tags. Similarly, the event handling will only trigger navigation events to such items.

5.2 Time Slider

We implemented a bidirectional time slider to specify the "relevant" time window for temporal filtering. This is shown in Figure 5.4.

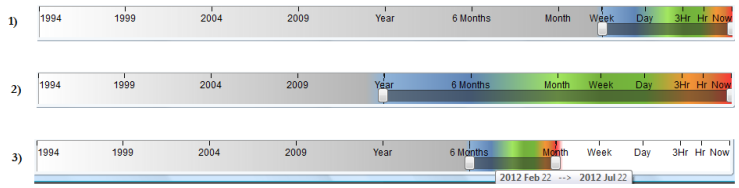

Figure 5.4: *Bidirectional time slider for specifying temporal filtering window. 1) In its initial state, it only includes files which have been modified over the past week, 2) After stretching one of the handles (they can be moved independently by clicking and dragging) to include all files that have been modified over the past 12 months, 3) A narrower time window, with a tooltip (displayed when moving the range or hovering over it) shows the time range currently selected*

5.2.1 Time Scale

It uses a pseudo-logarithmic scale, where recent times can be selected with more accuracy than older values. This is based on the assumptions that recent events are likely to be more important for users to be able to accurately filter, while with older events it may not be as easy to remember the exact dates. A one week range was chosen to be sufficiently restrictive while acknowledging that users may be working on several projects a week, and may take a few days off from working on a certain project over this time.

Markers for relative time ranges over the past year are distributed so that each is positioned halfway between the previous marker and the end of the widget. For example, the Year marker is at 50% of the slider width, 6 Months is at 75%, and so on.

Markers for year ranges are chosen such that only there is at most one for every 5 years (unless the file system contains less than five year's worth of files).

5.2.2 Colours

The colours show correspond to those used in SCOFT for indicating the newness of an item. Although several other colour schemes were tried, we found that target ages can be discerned most accurately when using this for recently visited items. Having said that, the current colouring does not work that well when the time range is greater than a year, as then most targets become red. This is acceptable when dealing with recent targets (where red is a nice eye catching colour), but does present problems for other time ranges.

5.3 Additional Techniques

We also developed a number of assistive techniques for supporting SCOFT. These are easy to deploy extensions to standard file browser components that aim to establish a connection between these components and SCOFT. This is intended to make it easier for users to infer the relationship between the different components, while providing additional spatially consistent tools for supporting revisitation behaviours.

5.3.1 Icon Highlights for Temporal Filtering

This technique is similar to the Icon Highlights technique discussed in [20].

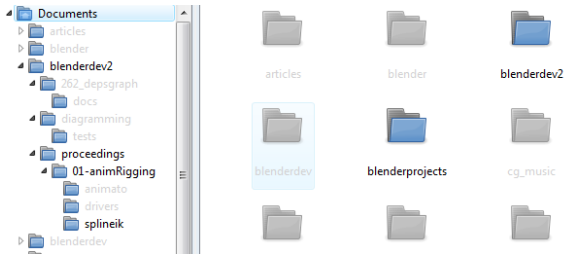

Figure 5.5: *Icon highlighting to support revisitation – 1) From the tree shown here, we can see that the relevance status gets propagated up the hierarchy to guide users to items of interest nested deep within the hierarchy. 2) Relevant items are highlighted to make it easier to identify targets on each level*

In order to support easier revisitation of items, we use a crude version of this technique to visually suppress (i.e. gray out) all items which do not fall within the range specified using the time range slider. This aims to improve user performance when searching for files by subconsciously drawing their attention to items within the specified time range (i.e. over the past week), which reduces visual search time required. We also propagate this *relevance status* up the hierarchy so that users can be guided to items of interest within their folders by following a series of highlighted folders.

As suggested by Fitchett [20], we only highlight folders where files have actually been opened (or modified) within the specified time range. This prevents casual/accidental browsing errors from cluttering the view.

5.3.2 Hierarchy Flattening

In our file crawler study (see Chapter 3), we found that many of the file system snapshots contained situations where a cluster of files and folders were nested at the tips of folder chains.

This situation is most common in Java projects, as these use deeply nested packages (e.g. `com.companyname.productname.modulename`) to avoid namespace conflicts. Since Java uses a file system based mapping scheme where classes and packages are mapped to files and folders respectively, source code files are usually stored at the tips of folder chains (i.e. zero-order hierarchies). Each these chains must be expanded on a level by level basis before users can access their files. However, it appears

that programmers often think of these namespaces as monolithic entities, as can be seen in the "Project Explorer" view, which flattens nested folder chains into Java styled package names [22].

We implemented two related mechanisms for improving navigation in this case. These altered the behaviour of the interface without changing the presentation of the structure to avoid disrupting users' spatial memory.

Automatic Tree Expansion

Firstly, we automatically expand such folders in the tree view so that users do not have to expand these themselves. Target folders at the tips of chains are only opened if they do not contain any sub-folders in order to prevent problems with target folders with a high branching factor (i.e. many sub-folders). In this case, we want to maintain some of the benefits of the collapsible approach, such as information hiding for large subtrees which are not currently relevant, so users would still need to explicitly expand such target folders. For example, in Figure 5.5, the three child folders of "blenderdev2" would be automatically expanded (as none of these have direct children, and they only contain a single sub-directory) but "01-animRigging" would not be expanded as it has several sub-folders.

Jumping Shortcut

The other technique was to jump to the tips of folder chains when opening folders at the root of or within such chains in the icon view. This acts as a shortcut for reducing the number of levels traversed (one of three design principles identified for improving file navigation times [20]). It is based on the assumption that users consider these chains as monolithic entities, and that the default action for interacting with these is just to navigate to the tip of the chain to access the files there. In the event that users wanted to modify part of the chain instead (for example, adding a new sub-module on one level), it is still possible to use the "Up One Level" functionality or the Folder Tree to navigate to chain members.

Algorithm

We deem a folder as being part of a chain if it only has a single child folder and contains "no files".

However, care must be taken with the "no files" clause, as some hierarchies may contain or require small configuration files at each level so that processing tools can recognise the folders. Examples of these files include the "`.directory`" file used by the Dolphin File Browser on KDE and "`__init__.py`" files used by Python to indicate modules. This problem was not really an issue in our implementation, as all system and hidden files (i.e. those whose names begin with a period) are not included in the datasets collected using the file crawler. For other implementations, we would recommend simply ignoring known system files by blacklisting those name patterns.

5.3.3 Crabbing

The final technique we developed allows users to navigate between folders at the same depth in parallel hierarchies, and also between sibling folders. We augment the standard icon view with two buttons on either side of the view (similar to those found in many image gallery viewers) which can be used to access the next and previous siblings. Figure 5.6 shows the different cases.

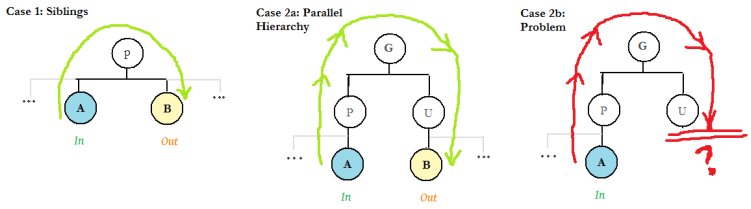

Figure 5.6: *The different cases which need to be handled for crabbing. 1) Between siblings, 2) Between parallel hierarchies, 3) Problematic situation, as may confuse user. Currently implementation accepts U, but it may be better to skip or stop.*

6 Evaluation

In this project, we aimed to improve file navigation by developing tools for supporting revisitation behaviours by reducing the number of navigation steps (clicks) needed to reach targets. In particular, we identified a few common usage scenarios such as:

- Navigating to frequently accessed files or projects
- Finding newly downloaded files or files modified during a certain time period
- Accessing source code files nested deeply within chains of source code trees
- Individually modifying and resaving many files from one directory into a parallel hierarchy

Our system has several mechanisms for supporting these use cases. The primary component is a spatially consistent representation of the entire file system, where items of interest are highlighted as a set of colour-coded tags superimposed over the file system visualisation. A bidirectional time range slider is provided for performing temporal filtering, where items which haven't been modified or accessed within this time range are greyed out to draw attention to relevant items. Folder chains are automatically expanded to remove the need to navigate or expand these chains one level at a time. Finally, buttons are provided on the edges of the icon view to make it easier to jump between parallel hierarchies.

To evaluate the effectiveness of these techniques, we performed a user study under controlled settings where participants were asked to perform representative tasks using our system and traditional systems. Here effectiveness was defined in terms of improved user performance (such as reduced task times, task completion rates, and error rates), but also in terms of qualitative measures (such as subjective satisfaction, and perceived effort and frustration involved in using the system).

We were interested in understanding the relative performance of this combination of techniques when used for a range of common use cases. This allowed us to validate whether the techniques are helpful in practice (for external validity), while helping to identify cases that may warrant further investigation and performance characterisation (for greater internal validity of important aspects).

6.1 Key Question

The primary question we aimed to address in our experiment was:

> Can "knowledge workers" (i.e. frequent computer users) use a spatially consistent representation of a large dataset (such as a file system containing thousands of entries) to access target locations more efficiently than using traditional interfaces (such as Windows Explorer) for hierarchy navigation?

Here, "target locations" is dependent upon the usage context, involving variables such as the user's current goals, work patterns, and to certain extent, the structure of the file system they're interacting with. In order to maintain some internal validity in our experiment, users are provided with the target locations they need to navigate to. These artificial targets were chosen to be representative of typical revisitation targets found in the use cases above.

6.2 Hypotheses

1. Users can navigate to frequently/recently used locations using SCOFT more efficiently than using traditional file browsers
2. Users can navigate to deeply nested targets more efficiently than using traditional file browsers
3. The temporal filtering facilities provided help users find targets more efficiently

6.3 Experiment Design

Participants were asked to perform a number of representative tasks with our interface and a version of the same software with our "enhancements" removed so that it resembled a traditional file browser. Different file hierarchy snapshots were used for each task to prevent cross-task learning effects and to prevent bias from certain datasets being better for certain interfaces.

For each task, a $2 \times n$ ANOVA within-subjects design was used, where n is the number of times target folders were visited during the experiment. This was 5 for Task 1, and 6 for Task 2. The factors studied were:

$$\begin{aligned} \text{Interface Type} &\in \{\text{Normal, Augmented}\} \\ \text{Visit Number} &\in \{1, 2, \ldots, n\} \end{aligned}$$

6.3.1 Participants and Apparatus

Participants were mostly post-graduate computer science students recruited from the Department of Computer Science and Software Engineering. There were 9 participants (1 female).

At the start of the experiment, participants were asked what operating system and file browsers they used most frequently or were most familiar with. These statistics are presented in Table 6.1.

OS	File Browser	Configuration	Number of Participants
Windows	Explorer	7	4
		Vista	2
Mac	Finder	Columns/Multi-Pane	2
		List/Details	1

Table 6.1: *Operating system and file browser experience statistics of experiment participants*

Some participants also reported some additional file browsing preferences and experience:

- 2 participants frequently used Linux and Nautilus in addition to Windows Vista and Explorer
- 1 participant made heavy use of keyboard shortcuts and type-ahead navigation
- 1 participant frequently accessed file systems on using the command line
- 1 participant configured Explorer in Windows 7 to display the folder tree by default (it is entirely hidden from view by default)

6.3.2 Apparatus Setup

The experiment was run on a Linux desktop running the KDE window manager, and displayed on a dual monitor setup. File browsing interfaces used in this study had dimensions of 900 x 570 pixels, and were displayed on the primary monitor (display resolution 1680 x 1040). A task prompting window was displayed on the secondary monitor (positioned to the left of primary monitor). It displayed the target for each task, and had additional controls for selecting the experiment progress. A screenshot of this setup is shown in Figure 6.1.

6.3.3 File Systems Used

Three file system snapshots from different sources were used in this experiment. We used real-life file system snapshots captured using our file crawler (see Chapter 3), as it is difficult to generate datasets of sufficient size and complexity while retaining enough semantically relevant targets and landmarks to ensure that the tasks remained realistic.

Two file systems were used for Task 1. Both were used by all participants (i.e. one interface used one file system, and the other interface used the other file system). These were chosen such that they both had similar numbers of folders (4000). To avoid bias effects, file systems weren't always paired with the same interfaces.

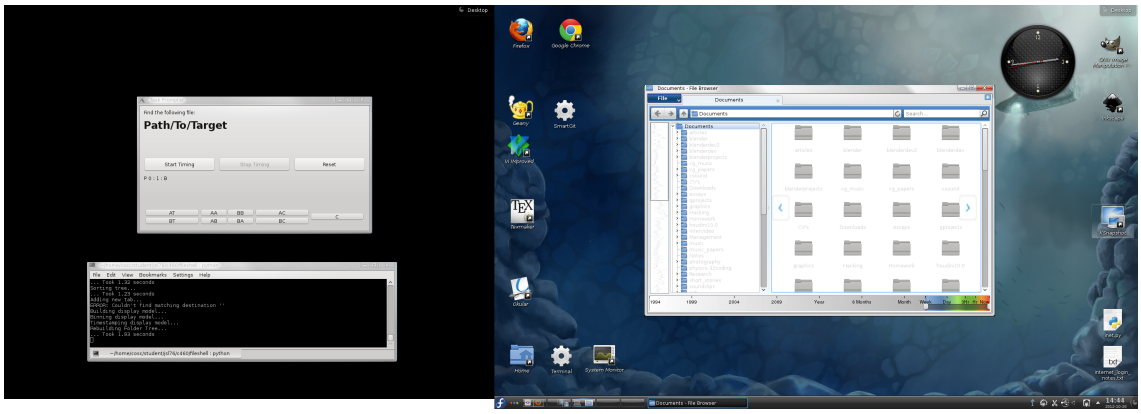

Figure 6.1: *Screenshot of experimental setup. The task prompter is displayed on the left, and the file browser interface ("augmented" condition shown) on the right.*

A subtree of one of the datasets used for Task 1 was used for Task 2 as well. This subtree was in a part of the hierarchy that was not visited in Task 1, and exactly resembled the intended experimental setup. Despite the potential for ordering effects bias here, it was difficult to find another suitable setup in any of the other datasets.

In addition to these two file systems, a third snapshot was used for all training tasks. Given that the intention is only to familiarise participants with the interfaces, their capabilities, and the experimental procedure, the effect of potential learning effects here are negligible.

6.3.4 Data Collection (Logging)

It was not possible to easily log every low-level user action (such as mouse movements and exact click locations) made during the experiment. Adequately instrumenting our systems to support this would be been prohibitively complex, and would have generated more data than we could easily make use of.

Instead, we used a simpler set of measures which would be sufficient for our study objectives. Namely, we focussed on navigation events (i.e. when a user tries to open/explore a folder, or backtracks to a previous visited location along the search path), which could be readily obtained by attaching custom handlers to the "navigation model" in our system. From these events, we could calculate the time taken to reach the target, as well as other measures such as the error rate (i.e. the number of backtracking steps = number of "back button" and "up one level" events).

Participant behaviour was observed during the experiment sessions. Notes were made of any significant events (notably any audible signs of frustration, confusion, or errors) that occurred (cross referenced using the progress indicator displayed on the prompter). In addition to this, paper-based NASA Task Load Index [42] worksheets were used to provide subjective data for each task, and were deployed between tasks while the experience of working with a system was still fresh in participants' minds. This asked participants to provide scores (on a scale of 1 to 5, low to high, good to poor) for a number of workload factors such as temporal, physical, and mental demands, as well as their level of frustration and perceived level of performance.

6.4 Procedure

Participants were required to perform two tasks with both interfaces during the experiment. There were originally three tasks, but the third task (an unstructured temporal search observation task) was dropped after we found that the experiment was taking an unreasonably long time to complete (i.e. between 30 and 40 minutes in some cases).

Participants completed all tasks with one interface before repeating the process with the other interface. This was done to reduce context-switching problems which could cause problems such as confusing which features worked in which interfaces.

Training tasks were performed before each condition. These were performed using a smaller number

of targets, and were intended to allow participants to become familiar with the usage of the interfaces for a particular task and to clarify any confusion about the experimental procedure. Before the first training task for each interface, participants were given an short introduction to the interfaces and the usage of their features where they were encouraged to play around with tools until they felt comfortable that they understood how to use them.

The "real" task conditions followed the training tasks, and involved larger sets of targets and/or more revisitations. Following each (non-training) task condition, a NASA-TLX [42] worksheet was completed.

6.4.1 Task 1 – Project Revisitation

The first part of the experiment was aimed at investigating how well our system supports revisitation behaviours for performing "everyday" computing tasks. That is, we were interested in the use case where a user is working on multiple projects, and navigates to these destinations many times.

This was primarily intended to evaluate the effectiveness of the SCOFT widget for its intended "fast access to revisitation targets" design goal. However, some of the other techniques could also be indirectly evaluated during this process, given that SCOFT can only provided revisitation support for folders that have been visited already.

Revisitation targets were identified by studying the characteristics of tags that were highlighted in the SCOFT widgets for "past week" time periods in the file system snapshots we obtained (Section 3.2). There were about 6 targets ($\mu = 5.5$, $\sigma = 2.67$) displayed, with targets about 3 levels deep ($\mu = 2.8$, $\sigma = 0.84$).

Hence, participants were required to find and select specific files within 6 different target locations, as specified by the task prompter. Participants were required to find and select files instead of simply navigating to a folder. This was to prevent disorientation and confusion when the interface is reset upon successful completion of the task. Each target was visited 6 times. The order in which targets were presented to participants was randomised (i.e. participants did not visit each target in a fixed order for each trial, and participants didn't perform all 6 visits in rapid succession). Following each successful selection, the interface would reset to its default state. This is equivalent to launching a new file browser, after the previous file browser was closed (i.e. for Open/Save dialogs, or if returning to project later in the day after an interruption).

Participants performed this task with both interfaces, with a different file system used for each interface. The order in which interfaces and file systems were used was counterbalanced to avoid bias effects due to interface ordering and file system structure.

6.4.2 Task 2 – Ping-Pong Navigation

This part of the experiment aimed to investigate how well our interface supports "ping-pong" navigation between parallel hierarchies. This use case often occurs when users need to perform a number of manual operations on files in one directory, while saving the modified files in a parallel folder. Examples of such operations include retouching photos (e.g. "crop, convert to grayscale, and adjust contrast" for yearbook photos), and cleaning up 3D models for importing into another application.

We were interested in determining the suitablility of the our interface for this task. This was measured in terms of whether users could perform this sort of navigation more efficiently (i.e. with shorter task times). We were also interested in understanding whether this functionality was able to be understood and used by users.

For this task, participants were shown a file browser with "folder A" open already. From here, they are required to select a particular file, and proceed to navigate to "folder B" where they select a file there. The next target would be in "folder A" again. This cycle was repeated 8 times per interface (Normal, Augmented), resulting in 14 selections being performed.

6.4.3 Post-Experiment Interview

After all experimental tasks had been completed, participants were interviewed to obtain feedback about the interfaces and tasks they had just performed.

They were asked about which interface they preferred for each task, and what the reasons for this choice. Building on these responses, participants were asked about whether they would use each of the features presented here if these were available in a file browser.

6.5 Results

A Python script was used to process the log data and compute the necessary metrics for each trial, such as time taken and number of navigation errors made.

Task times were calculated as the time difference between start and end events for trials in the log data. These events were generated only after the interface reset process has been completed and immediately after the participant successfully "opened" the target.

6.5.1 Task 1 – Revisitation

Trials for the decoy targets were removed from this analysis. These targets were originally intended to make the target locations less predictable, and were located at a shallower location than the other targets. We removed these from the analysis to reduce the amount of noise in our data, as we found that these had shorter task times than other targets.

Figure 6.2 shows a comparison between the average task times using the Normal and Augmented interfaces as targets are repeatedly visited. It can seen that target selection times reduce as targets are repeatedly visited for both conditions.

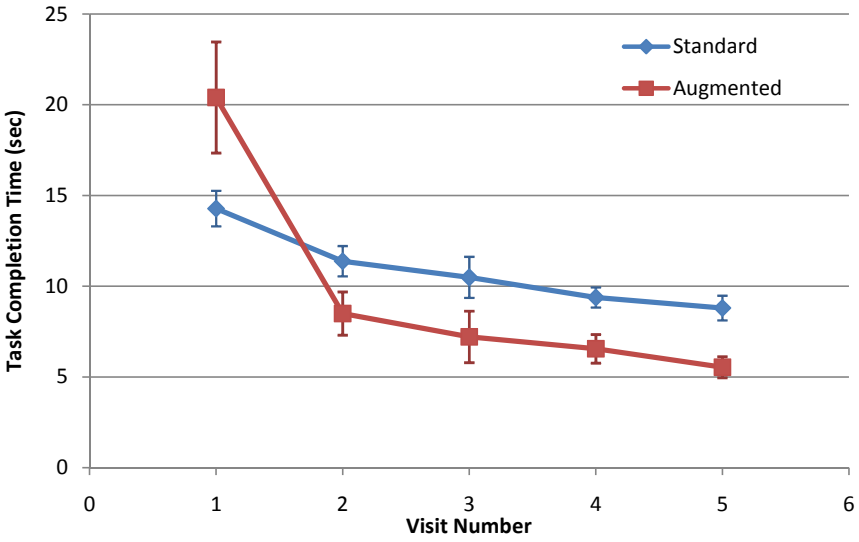

Figure 6.2: *Graph showing mean task times for repeated revisitiation of several different targets using* ***Normal*** *and* ***Augmented*** *file browsers. Error bars show ± 1 Standard Error.*

Initial Analysis – Full Results

We performed a 2×5 within-subjects ANOVA analysis of the results for this task (the factors were *Interface Type* and *Visit Number*).

There was no significant difference detected between the *Normal* and *Augmented* interface types, with mean task times of 10.866 sec and 9.638 sec ($F_{1,8} = 0.908$, $p = 0.369$). However, this appears to be caused by the large difference between the first and last task times for the *Augmented* condition (or more specifically, the high value of the initial task time), which raised the average task time and amount of variance in the data for the *Augmented* condition.

Significant effects were detected for repeated revisitation of targets. Mean task times decreased from 17.339 sec for the first visit, to 7.167 sec after 5 visits ($F_{4,32} = 24.19$, $p < .01$).

Significant interaction effects between the interfaces and repeated revisitation of targets were also detected. Mean task times for the *Normal* and *Augmented* interfaces decreased by 5.47 sec and 14.9 sec respectively ($F_{4,32} = 10.48$, $p < .01$). That is, participants were approximately 3 times more efficient when using the *Augmented* interface.

Analysis of Revisitation Tail

From the graph (Figure 6.2), participants appeared to be consistently more efficient using *Augmented* interface after they had visited a location at least once. We decided to re-analyse the data with the data points for the initial trial removed to focus on just this "steady state" period. This confirmed our visual analysis of the graph.

There was a significant difference detected between the *Normal* and *Augmented* interface types, with mean task times of 10.01 sec and 6.94 sec ($F_{1,8} = 5.540$, $p < .046$). This confirms that the *Augmented* interface is more efficient for targets that are frequently revisited.

There was also a significant difference detected for repeated revisitation of targets, with mean task times decreasing from 9.938 sec to 7.167 sec ($F_{3,24} = 6.346$, $p < .01$).

However, there were no significant interaction effects detected between interface type and repeated revisitation of targets ($F_{3,24} = 0.168$, $p = 0.917$). As can be seen from the graph, the user performance curves appear almost parallel. That is, the rate of change for both is approximately equal, which this test confirms.

NASA TLX

Figure 6.3 shows the results of NASA TLX worksheets filled out by participants following each condition to obtain subjective results.

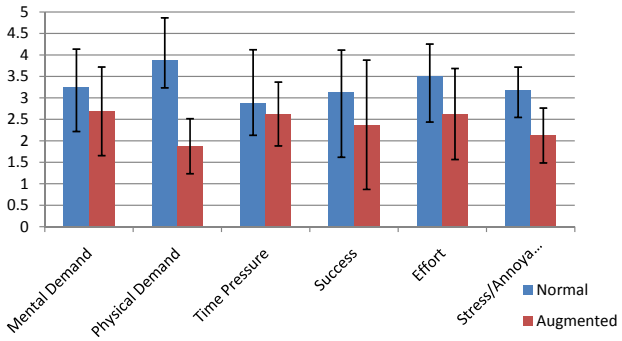

Figure 6.3: *Comparison of mean scores from Nasa TLX worksheets for Task 1. Lower values are better. Error bars show standard deviation.*

Subjective responses in all categories of the NASA TLX worksheets favoured the *Augmented* interface. The most significant differences were in terms of physical effort (fewer clicks required) and level of frustration.

6.5.2 Task 2 – Ping Pong Navigation

Many participants were confused about where they were in the hierarchy for the first two targets, and would end up spending time slowly retracing the hierarchy to figure out where they were. We have removed these trials from the results analysis, as these problems appear to affected only some of the participants.

Figure 6.4 compares the average task times taken for this task using the *Normal* and *Augmented* file browser interface conditions.

User Performance

We performed a 2×12 within-subjects ANOVA analysis of the results for this task (the factors were *Interface Type* and *Target Number*).

There was a significant effect for *Target Number*. Mean task times decreased from 3.746 sec to 2.957 sec ($F_{11,88} = 3.896$, $p < .01$). That is, navigation speed increased with revisitation frequency. Participants claimed that as they became familiar with the task, they could begin to anticipate the next target without needing to look at the prompter.

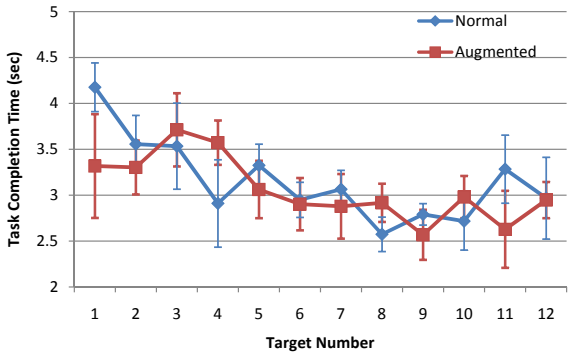

Figure 6.4: *Graph showing mean task times for ping-pong navigation between two folders using* ***Normal*** *and* ***Augmented*** *file browsers. Error bars show ± 1 Standard Error.*

No significant effects were detected for *Interface Type* ($F_{1,8} = 0.149$, $p = 0.710$). There were also no significant interactions between *Interface Type* and *Target Number* ($F_{11,88} = 1.429$, $p = 0.174$). This was because most participants used the same strategy of expanding the tree and clicking between the two destinations.

However, closer inspection of the graph reveals an interesting trend where the average task times for *Normal* and *Augmented* appeared to be somewhat complimentary.

NASA TLX

Figure 6.5 shows the results of NASA TLX [**?**] worksheets filled out by participants following each condition to obtain subjective results.

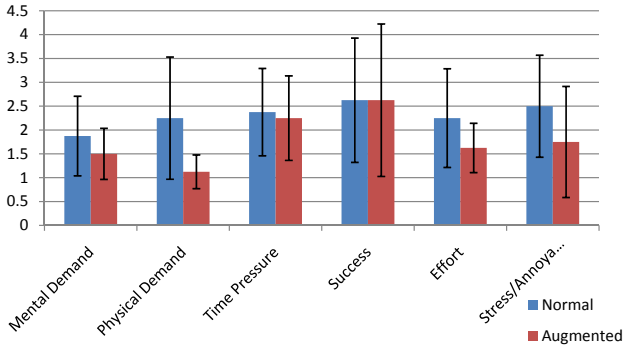

Figure 6.5: *Comparison of mean scores from Nasa TLX worksheets for Task 2. Lower values are better. Error bars show standard deviation.*

Subjective results favoured *Augmented* in most categories, and were equivalent to *Normal* in the remaining categories. The greatest difference was in terms of physical effort required, suggesting that the various techniques presented (including hierarchy flattening and crabbing) were somewhat effective at reducing the amount of clicks required. However, most participants felt that they had the same level of success using both interfaces, which would make sense if they used the same strategy in both interfaces.

7 Discussion

7.1 Key Findings of User Study

7.1.1 Task 1 – Revisitation

We found that the *Augmented* interface was more efficient for tasks where users need to repeatedly visit a number of targets in several different locations within a file hierarchy. Overall, mean task times for the *Normal* and *Augmented* interfaces decreased by 5.47 sec and 14.9 sec respectively ($F_{4,32} = 10.48$, $p < .01$), which means that participants were approximately 3 times more efficient when using the *Augmented* interface.

We also found that *Augmented* was consistently more efficient than the *Normal* for visiting targets that had already been visited recently (i.e. within the time range currently specified for temporal filtering). Mean task times for Normal and Augmented interfaces for such targets were 10.01 sec and 6.94 sec respectively ($F_{1,8} = 5.540$, $p < .046$).

However, these performance gains were relatively constant as targets were visited with increasing frequency, suggesting that there may have been other limiting factors involved. One possible explanation for this is the observation that participants did not appear to be able to correctly identify the right tags in SCOFT immediately for most trials. Instead, they often moused over or even clicked on 2 to 3 tags before finding the right one during the task. This error rate appeared to be lower during the practice task which only involved 3 target locations instead of 6. Further research is needed to better understand these performance characteristics.

Another notable finding was that *Augmented* was less efficient than the *Normal* for finding targets that had not been previously visited. The maximum times recorded for this task were 26 sec and 60 sec for *Normal* and *Augmented* interfaces respectively. This may be related to the temporal filtering feature, which would render all items which had not been touched within the time range using grey colors (including the labels). These findings suggest that this functionality may be somewhat detrimental to user performance when no revisitation data exists, possibly because it reduces the legibility of items (i.e. reduced text contrast and removal of colour-based cues when identifying icons). Perhaps controls to temporarily enable/disable temporal filtering may be necessary to solve this problem.

7.1.2 Task 2 – Ping Pong Navigation

The *Augmented* interface did not appear to show any net performance improvements over the *Normal* interface. This was primarily due to most participants using a single strategy across both interfaces which made only minimal use of the functionality provided.

When asked about why they performed the task in this way, most participants responded that it was simpler and/or faster to use the same strategy, with some participants saying that the task did not seem overly challenging.

7.2 Observations About Interface Usage

7.2.1 Tree-Based Hierarchy Navigation

It was surprising to see the most participants navigate the hierarchy using the folder tree. We had originally anticipated that participants would use the icon view to navigate through the hierarchy one level at a time. There were two factors which should have favoured that navigation style. Firstly, targets in the icon view were larger (i.e. they were 48×48 pixel icons in a grid with regular spacing instead of a tightly packed list of 16 pixel tall items). Secondly, file browser designs in the past few years have tended to remove these components in favour of simplified panels displaying a number of default system locations and user-defined bookmarks.

One possible explanation is that the way in which tasks were presented to participants encouraged these behaviours. Targets were presented as path strings (e.g. `460/reports/latexThesis/report.tex`). This may have encouraged users to use the tree, as it allowed them to maintain contextual awareness of where they were in the hierarchy. Furthermore, the expanded tree may have allowed participants to "scan ahead" and form a correspondence between elements in the path string and indented hierarchy elements. This theory may be supported by the observation that some users reported being distracted or even annoyed by the disorientation problems which occurred when a undetected bug in one of the experimental conditions would cause tree items to be scrolled to the top of the list when they were expanded or selected.

When asked about this behaviour, a common response was that there was less clutter when using the tree view. This allowed participants to quickly scan through the tree for folders more efficiently since only folders would appear there. Furthermore, selecting items in the tree would require less mouse movement. This meant that they would keep using the tree once they started using it since it was faster to keep using it.

One participant was observed alternating between the tree and the icon view (usually after going down two levels using the tree first). When asked why he would switch between the two, the participant said that this mainly occurred when the icon view displayed fewer items (i.e. less clutter), and would afterwards keep using it since the mouse was now over that part of the screen.

Another participant said that he used the tree out of habit. When probed further, he said that he had customised his copy of Windows Explorer to always display the folder tree. However, this does not appear to be a common behaviour.

Some caution is needed when interpreting these responses, especially as some participants admitted that they weren't sure why. This means that some of these responses may have merely been post-hoc rationalisations. Regardless of the underlying cause of this behaviour, we think it is worth questioning whether the current design decisions made by file browsers (of replacing the tree with a system links and bookmarks panel) are valid, especially with regard to facilitating efficient use of file systems.

7.2.2 Keyboard Usage

Three participants in this study asked whether they could use the keyboard to support faster navigation. Two of these were Mac users. These capabilities were not available in our prototype system primarily due to development resource constraints, but were also excluded in an attempt to reduce variance in our results which would have arisen if only a subgroup of participants used (and with great fluency from prior experience) certain techniques.

In post-experiment interviews, these participants expressed a desire to be able to use keyboard shortcuts for streamlining their interactions with the system. One interesting idea was to be able to use the TAB key to jump between tags identified by SCOFT, allowing even faster cycling between commonly visited locations.

7.2.3 Development of Spatial Memory

In most cases, there was a noticeable increase in navigation efficiency (but also reduced error rates) as participants visited certain target locations repeatedly. However, we noticed that all participants generally had some confusion about the exact location of targets within SCOFT, and would usually have to mouse over 2 different targets at different ends of the hierarchy space before making a selection.

7.3 Review of Study Design

7.3.1 Effectiveness and Realism of Experimental Tasks

The experimental tasks in this study were a compromise between having real-world tasks (with all the associated lack of internal validity), and slightly artificial versions that emulated some aspects of those tasks in a more controlled manner.

Task 1 – Revisitation

Subjects were required to navigate to 6 targets with equal frequency, and at equal depths in the file hierarchy. There was one decoy target, where item was located directly in the root folder, to act as a distractor to prevent participants from anticipating that a target could be accessed from that point.

For simplicity of analysis, we kept the number of revisitations per target constant instead of varying these by a Zipfian distribution. While it would have been even more realistic to have introduced such a distribution to reflect the variation of relative importance of revisitation targets, this would have reduce the internal validity of our results. Furthermore, in our experiment, we were aiming to obtain an initial estimate for the performance difference between the two different systems for "average" targets as they are revisited a number of times.

By making participants visit each target a number of times, we could gain a better estimate of how well participants would be able to cope with this usage scenario. The distribution of targets at heights in SCOFT also allowed us to also evaluate whether there were any negative effects between target locations and ability to repeatedly select a target.

Task 2 – Ping Pong

In retrospect, this task was somewhat artificial and less useful than originally expected.

It was originally intended to simulate the situation where users need to work on and resave a number of files from one folder into a parallel folder nearby. Our experimental setup aimed to reproduce this setup by having two parallel hierarchies: one folder with a number of files that needed to be accessed in turn, and another folder with just a single file. While it may have seemed that having participants repeatedly select that single file was somewhat artificial, we believe that this is an credible simulation of users using a "Save File" dialog from an application to save the contents of the first folder into the second folder (i.e. the single file in the second folder plays the role of the "Save" button of the dialog box here).

In particular, we had hoped that this task would have been able to demonstrate the effectiveness of the crabbing technique for navigation between folders in parallel hierarchies. However, a number of factors (namely incorrect initial assumptions) ended up confounding this role.

As mentioned earlier, we underestimated the tendency of participants to use the tree view. We found that once participants identified where they were in the tree at the start of the first trial (and overcame the related confusion), they could easily reorientate themselves relative to landmarks there, and proceed to use this to quickly jump between the folders. As one participant remarked, "it doesn't require much effort" to jump between the folders in parallel in this way, especially for targets just two levels deep. It was also reportedly "much faster" to use the tree instead of the left/right arrows as "system lag meant that it was more efficient to just move the mouse over to the tree and click a few times".

7.3.2 Apparatus

Interface Reset

After each target was selected, the interfaces were reset to their initial states in order to simulate standard file browser behaviour when opening new browser windows for each file browsing operation. A few participants remarked that this behaviour was "annoying" as it meant that they were forced to fully find and navigate to their targets again. However, we believe that this is an accurate representation of current file browser behaviour.

Task Times

Task times were calculated as the time elapsed between trial start and end events in the log data. Start events were generated automatically by the experiment apparatus once the interface reset procedure was completed. However, this does not account for the possibility that participants were distracted at the start of a task or perhaps took a short break between trials.

We could have accounted for such scenarios by cross-referencing our observation notes and removing ffending data points would been removed from the results. Even if we did not remove these data points, we would at least have been able to be identified as sources of error in our data.

An alternative solution would have been to use the first user action instead of the implicit trial start time (although this would end up introducing bias against situations where a participant may have had difficulties locating or deciding on an initial target). This was not done as we could not consistently apply this solution to our results, as the logging tool could not record the tree expand/collapse actions performed by participants in the first few trials. The apparatus needed to be modified before latter trials were conducted

to include logging support for these events, so that navigation statistics could be computed. These events were originally ignored as they were generated as a side effect of the hierarchy flattening implementation.

7.3.3 Sources of Experimental Error and Variance

System Lag

There was a slight lag between conditions for the Augmented condition which didn't exist for the Normal condition. This was caused by the recomputation of the filtering models when the interfaces were reset, as the filters needed to be recomputed for all files and folders (which would in turn need to flush the updated status to the corresponding underlying C++ widget components used by PyQt).

We attempted to account for this lag when logging trial start and end times by performing the interface reset before the timing period for each trial started. Although this avoids directly including these delays in our timing data, several participants reported that the delay affected their behaviour and performance in the experiment tasks. One problem they mentioned was that this caused uncertainty about whether they had finished the previous trial already, resulting in some hesitation to begin navigating in some cases. Another problem was that some functionality such as the Left/Right "Crabbing" arrows were not used participants perceived that moving the mouse over to the tree and clicking several times took less time overall despite requiring greater physical effort.

In retrospect, this was a confound in our experiment, as the Normal condition didn't have this lag. We could have reduced this problem ensuring that the filtering models were computed for both conditions, thus maintaining a consistent among of lag between both conditions (even though it didn't need to be present in one of these). It would have been even better to eliminate the lag altogether. However, that would have required significant re-engineering work for what is essentially still a proof-of-concept prototype.

Participant Speed

We observed varying levels of participant efficiency during the experiment sessions. Some were able to identify and acquire targets more quickly than others, while some appeared to be exercising great care with numerous cross-checks and careful cursor placement to expand the tree.

Our within-subjects experiment design meant that each participant acted as a control for their own variability, since they performed tasks with both interfaces. However, the variability in participant usage times did contribute to the experimental error in many cases. Having a larger sample size could have helped to smooth out this variance by adjusting the distribution weights.

7.4 Effectiveness of Solutions

Based on the findings of our experiment, we believe that the techniques that we have described in this report can be effective for some navigation tasks, and that users are willing to use them.

SCOFT

Participants were particularly enthusiastic about SCOFT and its support for quickly jumping between different target locations in a file system. All responded that they would use it for accessing projects they used frequently if it was included in a standard file browser.

Temporal Filtering

Participants were also supporting of the Icon Highlighting technique for performing temporal filtering. They found that it helped them identify targets more efficiently.

Crabbing

In general, participants were not too supportive of this technique. Many had difficulty understanding how it worked, or could not think of any situations where it could be used. One participant suggested that some of the problems may have been because the parallel hierarchy jumping was unpredictable, as it was difficult to understand where it may head to next, and that it may have been better if it simply cycled through the direct siblings only.

7.5 Similarities to Prior Work

The techniques presented here were directly inspired by several key techniques. These were:

SCOFT – Footprints [2], [15], and zoomed-out Fisheye DOI trees [56]

Temporal Filtering – FlexView [51], and Icon Highlights and Search Directed Navigation [20]

Hierarchy Flattening – Space Tree [45]

7.6 Limitations of SCOFT and Unresolved Issues

7.6.1 Large Folders

Large subtrees arising from software distribution bundles (such as the Java JDK) are problematic because they occupy a large portion of the SCOFT widget. This is because they often contain thousands of folders, which will therefore require more space in the visualisation. Such folders are rarely if ever accessed by users yet occupy a large amount of space while other more frequently visited targets become barely accessible.

As part of our characterisation work, we discovered that such folders could often be identified by searching for timestamps (within a 1 second range) where a large number of folders and files (for example, greater than 5 to 10) were created. If there are no subsequent access events for those items, that means that are large software distributions that have not been touched by users and are hence irrelevant.

7.6.2 Access to Files on Different Drives

It is currently not clear how file systems split across several different drives could be handled. As of the time of writing, we have experimented with manually stitching together datasets from relevant folders on different drives.

However, from our file crawler study, we found that some users need to access files on network-mounted drives. These present additional challenges as they may not always be available.

7.6.3 Synchronising File System Database

Currently the prototype implementation needs to maintain a copy of the entire file hierarchy in memory (read in from a backing Sqlite3 database) in order to draw the SCOFT widget and to correctly evaluate temporal filters. However, it is currently unknown how this representation could be kept in sync with the file system, especially when large numbers of files and folders are changed.

7.6.4 Processing Speed

As noticed during the experiment, there are some performance problems with the current prototype. This is to be expected with a Python based program which needs to operate on several thousand entities performing several time-range conversion operations on these every time some change occur. Compounding these problems is the underlying type conversions that must take place to flush updates from the internal system states to the C++ widgets used to render these in the user interface.

It is possible that additional performance could be obtained be rewriting time critical components in C++ and potentially multi-threading these computations as well. Multi-threading should be possible here for certain parts of the computations where there are no inter-object dependencies. However, due to the Global Interpreter Lock in Python, such optimisations are not possible.

7.7 Future Work

In addition to solving the limitations and unresolved issues, it is necessary to conduct more vigorous experiments to evaluate the effectiveness of this technique. For example, further work needs to be done to determine how many targets users can be shown before potential performance degradation occurs.

8 Conclusions

In this project, we aimed to investigate how a spatially consistent visualisation of file systems could be developed, which would facilitate easier and faster access to (frequently accessed) salient target locations.

To solve this problem, we developed a prototype file browser called FileShell. A primary component of this system was the SCOFT widget (**S**patially **CO**nsistent **F**older **T**humbnail), which displays a static visualisation of entire file systems.

We also developed a number of supporting techniques create a tighter integration between SCOFT and the file browser. One of these techniques was highlight-based temporal search, which helps support revisitation activities without distorting the spatial organisation of items by greying out items which did not match the relevance criteria. It can also be used to guide users towards targets they are interested in.

We drew on the results of an extensive literature review we performed to investigate of file browsing applications presented in prior literature in Human Computer Interaction. This covered work in sub-fields of this area such as file system paradigms, visualisations of hierarchical data sets, and efforts to understand the underlying human factors and the ways in which people use their file systems. Based on this review, we identified some design criteria and ideas for designing file navigation tools which would be able to overcome the limitations of prior attempts. For example, we avoided "radical innovation" in favour of adapting some more established techniques.

To evaluate our work, we conducted two user studies. In the first study, we ran a file crawler on several file systems to generate some datasets which we could investigate further. These were then used to validate our designs during the development process, and to further our understanding of the spatial and temporal characteristics of file systems.

A second user study was performed to evaluate the effectiveness of the techniques that we had developed. Here success meant that users are able to navigate to target locations more efficiently using the tools we have developed. We found that users could navigate to frequently visited locations 3 times faster than when using traditional file browsers for targets in a number of different locations, and were enthusiastic about being able to use similar tools in production file browsers.

In conclusion, we have successfully achieved our initial project goal of developing a spatially consistent file system visualisation technique which facilitates efficient revisitation of files, thus improving file navigation.

Bibliography

[1] The Places frecency algorithm. URL `https://developer.mozilla.org/en-US/docs/The_Places_frecency_algorithm`, 2011.

[2] Jason Alexander, Andy Cockburn, Stephen Fitchett, Carl Gutwin, and Saul Greenberg. Revisiting read wear: analysis, design, and evaluation of a footprints scrollbar. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1665–1674, New York, NY, USA, 2009. ACM.

[3] Benjamin B. Bederson, Ben Shneiderman, and Martin Wattenberg. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Trans. Graph.*, 21(4):833–854, October 2002.

[4] Ofer Bergman, Ruth Beyth-Marom, and Rafi Nachmias. The project fragmentation problem in personal information management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 271–274, New York, NY, USA, 2006. ACM.

[5] Ofer Bergman, Ruth Beyth-Marom, Rafi Nachmias, Noa Gradovitch, and Steve Whittaker. Improved search engines and navigation preference in personal information management. *ACM Trans. Inf. Syst.*, 26(4):20:1–20:24, October 2008.

[6] Ofer Bergman, Steve Whittaker, Mark Sanderson, Rafi Nachmias, and Anand Ramamoorthy. The effect of folder structure on personal file navigation. *J. Am. Soc. Inf. Sci. Technol.*, 61(12):2426–2441, December 2010.

[7] Ofer Bergman, Steve Whittaker, Mark Sanderson, Rafi Nachmias, and Anand Ramamoorthy. How do we find personal files?: the effect of OS, presentation & depth on file navigation. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 2977–2980, New York, NY, USA, 2012. ACM.

[8] Tristan Blanc-Brude and Dominique L. Scapin. What do people recall about their documents?: implications for desktop search tools. In *Proceedings of the 12th international conference on Intelligent user interfaces*, IUI '07, pages 102–111, New York, NY, USA, 2007. ACM.

[9] Richard Boardman. Bubble trees the visualization of hierarchical information structures. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '00, pages 315–316, New York, NY, USA, 2000. ACM.

[10] Richard Boardman and M. Angela Sasse. "Stuff goes into the computer and doesn't come out": a cross-tool study of personal information management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 583–590, New York, NY, USA, 2004. ACM.

[11] Richard Chimera and Ben Shneiderman. An Exploratory Evaluation of Three Interfaces for Browsing Large Hierarchical Tables of Contents. *ACM Transactions on Information Systems*, 12:383–406, 1994.

[12] Andy Cockburn and Carl Gutwin. A Predictive Model of Human Performance With Scrolling and Hierarchical Lists. *Human-Computer Interaction*, 24(3):273–314, 2009.

[13] Andy Cockburn, Carl Gutwin, and Jason Alexander. Faster document navigation with space-filling thumbnails. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 1–10, New York, NY, USA, 2006. ACM.

[14] Edward Cutrell, Susan T. Dumais, and Jaime Teevan. Searching to eliminate personal information management. *Commun. ACM*, 49(1):58–64, January 2006.

[15] Kushal Dave, Martin Wattenberg, and Michael Muller. Flash forums and forumReader: navigating a new kind of large-scale online discussion. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, pages 232–241, New York, NY, USA, 2004. ACM.

[16] Digia. Qt Development Framework. URL `qt.digia.com`, 2012.

[17] Susan Dumais, Edward Cutrell, JJ Cadiz, Gavin Jancke, Raman Sarin, and Daniel C. Robbins. Stuff I've seen: a system for personal information retrieval and re-use. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 72–79, New York, NY, USA, 2003. ACM.

[18] Facebook. Introducing Timeline. URL `https://www.facebook.com/about/timeline`, 2011.

[19] Jolon Faichney and Ruben Gonzalez. Goldleaf hierarchical document browser. In *Proceedings of the 2nd Australasian conference on User interface*, AUIC '01, pages 13–20, Washington, DC, USA, 2001. IEEE Computer Society.

[20] Stephen Fitchett and Andy Cockburn. Improving Navigation-Based File Retrieval. In submission to *CHI'13*.

[21] Stephen Fitchett and Andy Cockburn. AccessRank: Predicting what users will do next. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 2239–2242, New York, NY, USA, 2012. ACM.

[22] Eclipse Foundation. Eclipse IDE. URL `http://www.eclipse.org`, 2011.

[23] Eric Freeman. *The Lifestreams Software Architecture*. PhD thesis, Yale, 1997.

[24] George Furnas. Generalized Fisheye Views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '86, pages 16–23, New York, NY, USA, 1986. ACM.

[25] Google. Chrome Web Browser. URL `http://www.google.com/chrome`, 2012.

[26] Google. Storage Options — Android Developers. URL `http://developer.android.com/guide/topics/data/data-storage.html#filesExternal`, 2012.

[27] Onne Gorter. Database File System - An Alternative to Hierarchy Based File Systems. Master's thesis, University of Twente, 2004.

[28] Onne Gorter. DBFS - Database File System. URL `http://dbfs.sourceforge.net/`, 2004.

[29] Sarah Henderson. Genre, task, topic and time: facets of personal digital document management. In *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural*, CHINZ '05, pages 75–82, New York, NY, USA, 2005. ACM.

[30] Sarah Henderson. Document duplication: How users (struggle to) manage file copies and versions. *Proceedings of the American Society for Information Science and Technology*, 48(1):1–10, 2011.

[31] Sarah Henderson and Ananth Srinivasan. An Empirical Analysis of Personal Digital Document Structures. In *Proceedings of the Symposium on Human Interface 2009 on ConferenceUniversal Access in Human-Computer Interaction. Part I: Held as Part of HCI International 2009*, pages 394–403, Berlin, Heidelberg, 2009. Springer-Verlag.

[32] William C. Hill, James D. Hollan, Dave Wroblewski, and Tim McCandless. Edit wear and read wear. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 3–9, New York, NY, USA, 1992. ACM.

[33] Eleftherios Koutsofios Stephen North Gordon Woodhull John Ellson, Emden Gansner. Graphviz - Open Source Graph Drawing Tools. *Graph Drawing*, pages 483 – 484, 2001.

[34] Peter Pirolli John Lamping, Ramana Rao. A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. In *Proceedings of the ACM Conference on Human Factors and Computing Systems*, CHI '95, pages 401–408, 1995.

[35] Brian Johnson and Ben Shneiderman. Tree-Maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd conference on Visualization '91*, VIS '91, pages 284–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[36] Wolfgang Aigner Julius Penaranda, Jerome Penaranda and Silvia Miksch. Fisheye JTree. URL `http://ieg.ifs.tuwien.ac.at/projects/fisheye-jtree/`, 2006.

[37] Aparna Krishnan and Steve Jones. TimeSpace: activity-based temporal visualisation of personal information spaces. *Personal Ubiquitous Comput.*, 9(1):46–65, January 2005.

[38] Svenja Leifert. The influence of grids on spatial and content memory. In *PART 2 ——— Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, CHI EA '11, pages 941–946, New York, NY, USA, 2011. ACM.

[39] Clayton Lewis and John Rieman. *Task-Centered Interface Design: A Practical Introduction*. University Colorado, 1993. URL `http://www.acm.org/~perlman/uidesign.html`.

[40] LiquidFolders. LiquidFolders Windows Explorer Extension. URL `http://liquidfolders.net/start/en/`, 2012.

[41] Thomas W. Malone. How do people organize their desks?: Implications for the design of office information systems. *ACM Trans. Inf. Syst.*, 1(1):99–112, January 1983.

[42] NASA. Nasa Task Load Index. URL `http://humansystems.arc.nasa.gov/groups/TLX/`, 1988.

[43] Jakob Nielsen. Ten Usability Heuristics. URL `http://www.useit.com/papers/heuristic/heuristic_list.html`, 2005.

[44] Rob Pike. History of Dot Files. URL `https://plus.google.com/101960720994009339267/posts/R58WgWwN9jp`, 2012.

[45] Catherine Plaisant, Jesse Grosjean, and Benjamin B. Bederson. SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)*, INFOVIS '02, pages 57–, Washington, DC, USA, 2002. IEEE Computer Society.

[46] Jun Rekimoto. Time-machine computing: a time-centric approach for the information environment. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, UIST '99, pages 45–54, New York, NY, USA, 1999. ACM.

[47] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone Trees: animated 3D visualizations of hierarchical information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 189–194, New York, NY, USA, 1991. ACM.

[48] Michael Rooke, Tovi Grossman, and George Fitzmaurice. AppMap: exploring user interface visualizations. In *Proceedings of Graphics Interface 2011*, GI '11, pages 111–118, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2011. Canadian Human-Computer Communications Society.

[49] Joey Scarr and Andy Cockburn. Testing the Robustness and Performance of Spatially Consistent Interfaces. In submission to *CHI'13*.

[50] Joey Scarr, Andy Cockburn, Carl Gutwin, and Andrea Bunt. Improving command selection with CommandMaps. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 257–266, New York, NY, USA, 2012. ACM.

[51] Doug Schaffer and Saul Greenberg. Sifting through hierarchical information. In *INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems*, CHI '93, pages 173–174, New York, NY, USA, 1993. ACM.

[52] John Stasko. An evaluation of space-filling information visualizations for depicting hierarchical structures. *Int. J. Hum.-Comput. Stud.*, 53(5):663–694, November 2000.

[53] John Stasko and Eugene Zhang. Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations. In *Proceedings of the IEEE Symposium on Information Vizualization 2000*, INFOVIS '00, pages 57–, Washington, DC, USA, 2000. IEEE Computer Society.

[54] Susanne Tak, Joey Scarr, Carl Gutwin, and Andy Cockburn. Supporting window switching with spatially consistent thumbnail zones: design and evaluation. In *Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part I*, INTERACT'11, pages 331–347, Berlin, Heidelberg, 2011. Springer-Verlag.

[55] GraphViz Team. GraphViz Documentation. URL `http://www.graphviz.org/Documentation.php`, 2011.

[56] Christian Tominski, James Abello, Frank van Ham, and Heidrun Schumann. Fisheye Tree Views and Lenses for Graph Visualization. In *Proceedings of the conference on Information Visualization*, IV '06, pages 17–24, Washington, DC, USA, 2006. IEEE Computer Society.

[57] Wikipedia. Spatial File Browsers. URL `http://en.wikipedia.org/wiki/Spatial_file_manager`, 2012.

[58] Wikipedia. Zipf's Law. URL `http://en.wikipedia.org/wiki/Zipf%27s_law`, 2012.

[59] Wikipedia. Zoomable User Interfaces. URL `http://en.wikipedia.org/wiki/Zooming_user_interface`, 2012.

[60] Rajdeep Gill William Jones, Ammy Jiranida Phuwanartnurak and Harry Bruce. Don't take my folders away!: organizing personal information to get ghings done. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '05, pages 1505–1508, New York, NY, USA, 2005. ACM.